TREBALL DE FI DE CARRERA

TÍTULO: Redes mesh basadas en puntos de acceso inteligentes 802.11 open source (I)

AUTOR: Lluis Faixó Rubio

DIRECTOR: Eduard García Villegas

CODIRECTOR: Rafael Vidal Ferré

DATA: 5 de setembre de 2005

Título: Redes mesh basadas en puntos de acceso inteligentes 802.11 open source (I)

Autor: Lluis Faixó Rubio

Director: Eduard García Villegas

Codirector: Rafael Vidal Ferré

Data: 5 de setembre de 2005

Resumen

Este documento contiene los datos teóricos y prácticos, así como la metodología de trabajo seguida para el desarrollo del proyecto "Redes mesh basado en puntos de acceso inteligentes 802.11 open source (I)".

Este proyecto trata de dotar a ordenadores personales la capacidad de trabajar como APs de una red inalámbricas. Con la característica de que estos APs sean inteligentes y tengan la capacidad de elegir el canal en el que trabajan para poder dotar a la red de un mayor rendimiento. Estos APs trabajan con 802.11b, utilizando tarjetas inalámbricas PCMCIA. Cada ordenador cuenta con dos tarjetas, una se utilitzará en modo Ad-hoc para la comunicación entre los distintos APs y la retransmisión de la información que se transmita por la red. La otra interfaz trabaja en modo infraestructura, y es la encargada de dar cobertura a los clientes de la red.

La red trabaja con IPv6. Para que esta funcione y permita la conectividad entre todos los nodos necesita un protocolo de enrutamiento para redes inalámbricas ad-hoc. Lo primero que se ha realizado es un resumen de las características más importantes. Posteriormente se han analizado los protocolos más importantes para acabar eligiendo el que más se adecuaba a las necesidades de la red con la que se trabaja.

Se ha elegido el protocolo OLSR del que hay una buena implementación que es la que se ha utilizado y a la que se le han añadido funcionalidades para la distribución de información. La información que se transmite además de la referente a la propia topología de la red, es información de las interferencias creadas por los vecinos al nodo. También se debe conocer los canales en los que trabajan todos los nodos así como la utilitzación que está teniendo ese nodo.

Toda esta información se debe almacenar en un archivo, con un formato especial para ser posteriormente tratada y de este modo poder mejorar la eficiencia de la red.

Title: Networks mesh based on points of intelligent access 802.11 open source (I).

Author: Lluís Faixó Rubio

Director: Eduard García Villegas

Codirector: Rafael Vidal Ferré

Date: September, 5th 2005

Overview

This document contains the practical and theoretical data, as well as the methodology of continued work for the development of the project "Networks mesh based on intelligent access points 802.11 open source (I)".

This project tries to endow to personal computers the capacity to work as a AP's of a network wireless. With the characteristic that these APs must be intelligent and they have the capacity to elect the channel in which they work to be able to endow to the network of a greater performance. These AP's work with 802.11b, utilizing PCMCIA wireless cards. Each computer have two cards. The first one will be use in way Ad-hoc for the communication between the different AP's and the broadcast of the information that is transmitted for the network. The other interface works in infrastructure mode, and is the responsible for giving cover to the network's clients.

The network works with IPv6. To permit the connectivity among all the nodes needs a routing protocol for wireless ad-hoc networks. The first thing that has been carried out is a summary of the most important characteristics. Subsequently the most important protocols have been analyzed to finish choosing the one that more was adapted to the needs of the network with the one that works itself.

The protocol elected has been OLSR of the one that there is a good implementation that is the one that has been utilized and to which they have been added him functionalities for the distribution of information. The information that is transmitted besides the referring one to the own topology of the network, is information of the interferences created by the neighbours to the node. Also it should be known the channels in which all the nodes work as well as the troughput that is having that node.

All this information should be stored in a file, with a special format for subsequently to be treated and in this way being able to improve the efficiency of the network.

INTRODUCCIÓN

Las redes inalámbricas han experimentado un importante auge en los últimos años debido a la aparición de dispositivos basados en la serie de normas 802.11x, baratos y fáciles de utilizar, proporcionando una alternativa a las redes cableadas habituales. Las redes inalámbricas permiten una flexibilidad y movilidad al usuario sin tener que sacrificar la conexión a Internet o a la red informática del lugar de trabajo. Pero estas redes están evolucionando de manera vertiginosa, mejorando sus prestaciones mediante nuevos equipos, pero sobretodo gracias a nuevos tipos de redes con configuraciones físicas y lógicas distintas. Un ejemplo de estas son las redes malladas inalámbricas.

Las Inalámbricas Mesh Networks (WMNs) consisten en dos tipos de nodos los enrutadores y los clientes, donde los enrutadores tienen movilidad mínima y forman el backbone de las WMNs. Estas redes pueden integrarse mediante funciones gateway y de puente con redes como Internet, IEEE 802.11, IEEE 802.15, IEEE 802.16, etc. Los clientes pueden ser estáticos o móviles y pueden crear una red mallada entre ellos mismos o con los enrutadores. Estas redes solucionan las limitaciones y mejoran el rendimiento de las redes ad-hoc. Gracias a la posibilidad de conectarse a distintos puntos de acceso en lugar de a uno sólo se aumenta el ancho de banda que puede tener cada cliente, también resulta mucho más estable ya que puede seguir funcionando aunque caiga un nodo, en cambio en las redes habituales si caía un punto de acceso los usuarios de ese punto de acceso se quedaban sin servicio. En la siguiente figura se puede observar una red mallada ejemplo en la que se pueden ver las distintas características que pueden tener estas redes.

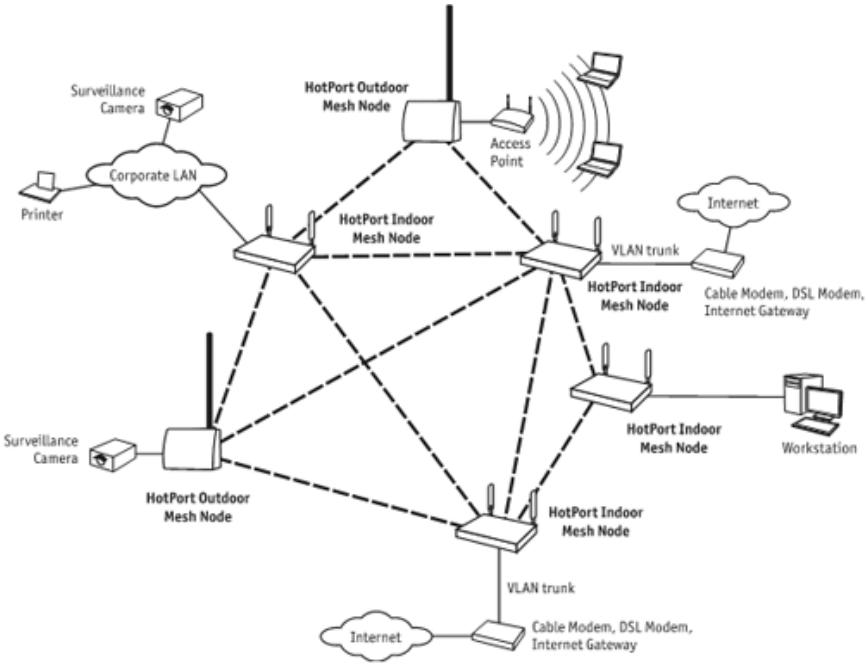


Fig 0.1 Ejemplo red mallada inalámbricas

Estas redes se organizan y se autoconfiguran automáticamente: entre los nodos de la red se establecen automáticamente los enlaces y mantiene la conectividad en malla entre ellos. Además, este proceso se realiza dinámicamente, permitiendo la aparición de nuevos nodos y la desaparición de nodos existentes. Estas características aportan grandes ventajas como robustez, fiabilidad y un mantenimiento fácil de redes. Para obtener una mejor flexibilidad de las redes malladas, los nodos están normalmente equipados con múltiples interfaces que pueden ser de diferentes tecnologías de acceso inalámbricas.

En la actualidad hay una numerosos *testbeds* de WMN. Los ejemplos más destacados los podemos encontrar en la Universidad Carnegie-Mellon[1] y en la de Illinois. En la primera, el testbed consta de siete nodos, 2 estáticos y otros 5 montados en coches que circulan por todo el campus. Los paquetes se enrutan mediante el protocolo DSR (Dynamic Source Routing) y han experimentado con diferentes tráficos incluso con stream de audio y video. En la universidad de Illinois tienen una red multicanal de 4 nodos en la que han conseguido un throughput 2.63 veces superior a una red de un solo canal.

Esta configuración radio (o inalámbricas) se debe hacer teniendo en cuenta los canales en los que deben trabajar los nodos de la red, puesto que en caso contrario la señal, debido a las interferencias, sufre un gran deterioro y no se obtiene ninguna mejora en el rendimiento de la red. Este deterioro es debido a que el medio de acceso del 802.11 utiliza el protocolo CSMA/CA con reconocimiento y un tiempo de *back off* aleatorio, este protocolo se diseñó para reducir la probabilidad de colisión entre los múltiples intentos de acceso al medio. Las múltiples estaciones están esperando que el medio esté libre y cuando lo está todas las estaciones intentan acceder al mismo tiempo. Por lo tanto se utiliza una distribución *back off* aleatoria para poder minimizar los conflictos de medio. Esto hace que el ancho de banda decrezca si todos los nodos utilizan el mismo canal debido a que se deben realizar múltiples retransmisiones. Además para tener una comunicación libre sin interferencias de canales vecinos las comunicaciones se deben hacer con una distancia de 5 canales, es decir, si un nodo trabaja en el canal 1 el vecino debe trabajar en el 6 para no tener ningún tipo de interferencia. Por este motivo la selección de canal es un aspecto muy importante en la mejora del rendimiento de este tipo de redes.

En este proyecto se trata de dotar de inteligencia a los APs, para que tengan la capacidad de colaborar para elegir de manera distribuida la asignación de canales que minimiza las interferencias. Para ello se trabajará con ordenadores personales que trabajarán con el sistema operativo Linux, y el conocido driver Hostap para que puedan trabajar como puntos de acceso. La mayoría de las redes inalámbricas son del tipo infraestructura, pero en este caso se crea una red mallada para comunicar los diferentes APs entre sí mediante el uso de una seguna interfaz inalámbrica: una se usa para dar servicio a sus clientes y la otra para formar parte de una red ad-hoc que puede ser usada tanto para el intercambio de información de señalización, como para funcionar como red

[1] http://www.cmu.edu/computing/wireless/

troncal inalámbrica. Otra ventaja es que si se quiere unir un nuevo AP a la red no es necesario configurar manualmente la red, ni crear una infraestructura debido a que son autoconfigurables y sólo necesita una toma de corriente para acceder a todos los servicios.

El primer objetivo del proyecto es el de crear una red inteligente donde los nodos sean capaces de elegir el canal en el que trabajan. Como segundo objetivo está el de dar la capacidad a toda la red de trabajar con IPv6, por lo que se deberá configurar los sistemas operativos para que puedan trabajar con este protocolo. Como tercer objetivo se encuentra dotar de la inteligencia necesaria para que los nodos puedan crear rutas dinámicas hacia los otros nodos de la red, además de ser capaces de descubrir los nodos que aparezcan durante el funcionamiento. Para ello se debe dotar a estos ordenadores de un protocolo de enrutamiento pensado para redes inalámbricas. Como último objetivo se deberá adaptar este protocolo a la red que se pretende crear y además añadirle funcionalidades para que cumpla con todas las características que se le demanden.

En el primer capítulo se dará una visión general de las propiedades que tiene los protocolos de enrutamiento inalámbricas y posteriormente se explicarán en detalle algunos de los protocolos más utilizados y que tienen una implementación más cercana a la definición teórica.

En el segundo capítulo se procederá a la explicación de la maqueta de la red. Empezando con la descripción del hardware y software utilizado, la instalación de todos los componentes software necesarios para tener los nodos adecuados, el protocolo de enrutamiento, capacidad IPv6 y finalmente se procederá a la configuración de la maqueta y las IPs que se le dan a todos los nodos. Además de esto también se explicará la aplicación desarrollado para la selección de un canal que no cree interferencias en los nodos actuales cuando arranca un nuevo nodo.

En el tercer capítulo se explican con más detalle aspectos que se deben variar para el buen funcionamiento del protocolo de enrutamiento. Como son el cambio de algunas variables y el añadido de un plugin con el que se consigue que los nodos se intercambien información valiosa para nuestra red. Esta información consiste en los valores de potencia recibidos de otros nodos vecinos, es decir, la interferencia que reciben unos nodos de otros vecinos. Estos datos serán utilizados para poder seleccionar el canal en el que deben trabajar lo nodos para que el rendimiento de la red sea óptimo.

Una vez creada y configurada la red con todos los detalles, se procederá al desarrollo de las pruebas así como a dar distintas infraestructuras a la red, es decir, distintos números de nodos y ubicación de ellos.

En el último apartado del presente documento se comentan las conclusiones a las que se ha llegado durante el desarrollo del proyecto y las posibles líneas futuras que se podrían seguir para continuar con este proyecto.

Después de esto sólo quedan dos apartados que se adjuntarán como anexos. En el primero de ellos se puede ver de manera detallada el script de selección de canal. En el último apartado se encuentra el código fuente del plugin que se ha desarrollado para el protocolo OLSR.

Capítulo 1. Enrutamiento

1.1 Clasificación

En este apartado se pretende dar una breve introducción a los protocolos de enrutamiento en redes Inalámbricas Ad Hoc. Se dará una pequeña clasificación de sus características y tipos más conocidos.

Estas redes como se sabe no dependen de ninguna infraestructura física por lo que la comunicación viene dada por los sistemas de radio de los equipos. Para que se produzca esta comunicación entre los equipos, estos deben trabajar como routers, pero este enrutamiento es mucho más complejo que en redes fijas. Este problema se intenta resolver mediante un gran número de algoritmos y protocolos de enrutamiento.

Hay un gran número de algoritmos por que esta tecnología está aún en investigación y se buscan soluciones desde vías muy diferentes, lo que da un gran número de protocolos, los cuales se definen con características y políticas muy diferentes.

Lo que se pretende en los siguientes párrafos es indicar las características más representativas y compararlas con sus antagónicas. Cabe mencionar que no todas las filosofías estudiadas están reñidas; aunque diferentes protocolos tengan una característica principal muy diferente no siempre significa que su procedimiento sea muy distinto.

1.1.1 Canal único/Multicanal

Esta es una propiedad de capa 2. Hay protocolos en los que todos los nodos comparten el mismo canal de comunicación. Lo que significa que todas las comunicaciones pasan por el mismo canal disminuyendo la velocidad de la red.

El medio de acceso del 802.11 utiliza el protocolo CSMA/CA con reconocimiento y un tiempo de *back off* aleatorio que sigue una condición de medio ocupado. El protocolo CSMA/CA de 802.11 se diseñó para reducir la probabilidad de colisión entre los múltiples intentos de acceso al medio. Las múltiples estaciones están esperando que el medio este libre y cuando lo está todas las estaciones intentan acceder al mismo tiempo. Por lo tanto se utiliza una distribución *back off* aleatoria para poder minimizar los conflictos de medio.

Hay protocolos que especifican el canal de comunicación como: ADV, AODV, OLSR, etc. Pero en cambio hay otros protocolos que no especifican el canal de comunicación como: ABR, CBRP, DREAM…

En cambio, otros protocolos utilizan CDMA, FDMA o TDMA para poder especificar el canal. En este caso la comunicación es mucho más eficiente

porque se puede trabajar con velocidades más altas pero en contra se tiene que controlar mediante las estaciones la asignación de canales. Los protocolos que siguen esta especificación son: TORA, CGSR, TLR/TRR.

1.1.2 Uniforme/No uniforme

Esta característica básicamente nos permite indicar si todos los nodos de la red tienen las mismas especificaciones, características y rol, en cuyo caso es uniforme. Un protocolo no uniforme indica que hay nodos de la red con características distintas, lo cual significa que hay nodos con roles distintos.

Estos roles pueden venir dados por el hecho de que la red esté dividida en diferentes clusters y hay un nodo de este cluster que es el principal o incluso hay un nodo gateway. Ejemplos de este tipo de protocolos son CBRP, CGSR, CEDAR, DST, HSR, LANMAR, OLSR.

1.1.3 Topología jerárquica/Enrutamiento con Clusters

La idea de utilizar clusters en las redes Ad Hoc es para intentar dar una estructura a este tipo de redes. Estos clusters habitualmente tienen un nodo principal dedicado, que es el encargado de indicar a los nodos cercanos a qué cluster pertenecen y de este modo estructurar la red. Estos nodos principales a parte de ser informados de la unión o desunión de nodos también se encargan de ser las puertas de unión entre los diferentes clusters. Se debe mencionar que en muchos casos los clusters tienen diferentes capas jerárquicas, en este tipo de protocolos se envía información más frecuentemente a los nodos que se mueven rápidamente o a los que están más cerca.

El problema de los clusters es que el nodo principal y el gateway tienen que trabajar con mucha información y se pueden convertir en el cuello de botella de la red, ya que si todos los nodos de la red pretenden enviar información a otro nodo, toda esta información tratará de salir por el mismo nodo. Además de ralentizar la comunicación, estos nodos también gastarán mucha más energía que los otros, lo que puede suponer un problema en nodos que trabajen con fuentes autónomas de alimentación.

Los protocolos que trabajan con clustering o estructura jerárquica son los siguientes: CBRP, CGSR, FSLS, FSR, HSR, LANMAR y ZRP.

1.1.4 Protocolos basados en posicionamiento

Estos protocolos se basan en la idea de que no es necesario mantener tablas de enrutamiento y por lo tanto no es necesario el overhead debido al mecanismo de descubrimiento o mantenimiento de rutas. En cambio, necesitan

saber la posición de los destinatarios de la información, ya sea mediante un protocolo interno de descubrimiento, o por un servicio de localización externo que obligará a mantener información de posicionamiento.

Otro tipo de protocolos son los que se basan en inundación, en los que se debe conocer aproximadamente la zona del nodo destino y entonces se envían los paquetes a todos los nodos que estén en la dirección de la zona destinataria, de este modo no se tienen que enviar los mensajes a todos los nodos vecinos.

Cabe mencionar que hay protocolos híbridos que para distancias largas utilizan enrutamiento direccional, es decir, se basan en conocer la dirección en la que se encuentra el nodo y transmitir en esa dirección. Pero para los nodos cercanos no se utiliza ningún mecanismo basado en localización.

Los protocolos basados en posicionamiento son los siguientes: DREAM, GEDIR, GPSR, LAR y Terminode/AGPF.

1.1.5 Proactivo/Enrutamiento bajo demanda

Un protocolo de enrutamiento puede mantener la información bajo demanda (reactiva), es decir, actualiza su información de enrutamiento a medida que es necesaria. Este tipo de protocolos no necesita que todos los nodos tengan la información de enrutamiento en todo momento, sino que la actualizarán a medida que la necesitan. Lo que se pretende conseguir es que la red inalámbrica no tenga una gran carga de señalización innecesaria. Se puede considerar muy útil cuando la información viaja a menudo por rutas muy parecidas.

Estos protocolos necesitan saber al menos el primer salto que deben hacer, si no lo conocen se debe hacer un broadcast hacia todos los nodos vecinos, esta estrategia sólo se puede utilizar en los primeros saltos, si se utilizara en exceso se inundaría la red, lo que no es conveniente. Los paquetes no se empiezan a enviar hasta que la ruta no está especificada, esto supone un retraso en el envío de los primeros paquetes. Una vez la ruta está finalizada, se debe guardar en caché la tabla de enrutamiento durante un periodo de tiempo, una vez pasa este tiempo la ruta se invalida.

Los protocolos proactivos, al contrario que los reactivos (bajo demanda), intentan mantener toda la información de enrutamiento correcta en todos los nodos de la red en cada momento. Estos protocolos también se pueden dividir en dos clases: los que tratan eventos y los que se actualizan de manera regular.

Los que trabajan con eventos no envían paquetes de actualización hasta que no hay un cambio en la topología de la red. En cambio, en el caso de actualización regular, la información se retransmite cada cierto tiempo. La ventaja de este tipo de protocolos es que no necesitan un tiempo para crear la ruta, por el contrario añaden mucha más carga a la red.

También hay protocolos híbridos en los que se unen los dos tipos, lo que significa que las rutas se mantienen de manera proactiva pero sólo en ciertos nodos, no en todos. Ejemplos de estos protocolos son ADV, ZRP y Terminode Routing.

1.1.6 Información de topología reducida/completa

Esta división es muy difícil de evaluar porque, aunque la mayoría de protocolos transmiten información de la topología de la red, no todos transmiten información completa sobre la estructura de la red. Protocolos que tienen toda la información de la topología son: DDR, GSR… en el otro lado, donde la topología es reducida es: FSLS, FSR, LANMAR...

1.1.6.1 Uso del enrutamiento fuente

Algunos protocolos de enrutamiento utilizan enrutamiento de fuente, esto significa que el envío depende de la fuente, es decir, que la fuente pone la información en la cabecera de la información de enrutamiento. Los nodos siguientes utilizan la información de la cabecera para el reenvío. CBRP, DSR, Terminode/AGPF y WAR son protocolos de este tipo.

1.1.6.2 Uso de los mensajes de broadcast

Los mensajes Broadcast en una red inalámbricas tiene distintos significados, están los "full netwide broadcast" lo que significa que el mensajes llega a todos los nodos de la red y es necesario que algunos sean retransmitidos por nodos intermedios. Por el contrario están los "local broadcast" que envía los mensajes a todos los nodos vecinos, no se hacen retransmisiones. También se debe mencionar que algunas veces también hay broadcast limitados en los que se establece un número máximo de saltos (TTL).

1.1.7 Estrategia de selección de ruta

Este es uno de los aspectos más importantes de los protocolos de encaminamiento obviamente, hay que tener en cuenta que puede haber muchas maneras de elegir la estrategia. Las más conocidas y utilizadas son las siguientes:

- Signal Strength: encamina los paquetes a través de la conexión siguiendo los que tienen la señal más fuerte.
- Estabilidad de conexión: los paquetes viajan por los nodos que parecen

más estables durante un periodo de tiempo.
- Camino más corto/Estado de la conexión: Selecciona el camino más corto siguiendo algunas métricas como la calidad del enlace.
- Vector de distancia: método de vector de distancia común, normalmente basado en la cuenta de saltos.
- Encaminamiento direccional: encamina hacia la dirección geográfica del destinatario, usado en los protocolos de localización.

1.2 AODV (Ad hoc On-Demand Distance Vector)

Este protocolo permite el enrutamiento dinámico, autoarranque y *multihop* entre todos los nodos móviles que participan en la red. AODV permite a todos los nodos obtener las rutas rápidamente para las nuevas destinaciones y no requiere que los nodos mantengan las rutas hacia los destinos que no están activos en la comunicación.

El protocolo de enrutamiento está diseñado para redes móviles ad hoc con gran cantidad de nodos y con distintos grados de movilidad. Este protocolo se basa en que todos los nodos tienen que confiar en los otros para transportar sus datos, aunque sea por el uso de una clave preconfigurada, o activando mecanismos para evitar la participación de nodos intrusos.

En este apartado lo que se intenta es dar una breve introducción de sus características y sus modos de funcionamiento básicos, así como sus tablas y mensajes más característicos sin entrar en el formato de estos.

Una característica distintiva de este protocolo es el uso del número de secuencia para cada ruta. Este número de secuencia es creado por el destino para ser incluido con la información necesaria para los nodos que requieren la información. El uso de estos números implica que no se crean *bucles* y la facilidad de programación.

Este protocolo define tres tipos de mensajes: Route Requests (RREQs), Route Replies (RREPs) y Route Errors (RERRs). Estos mensajes se reciben vía UDP. Mientras todos los nodos tengan las rutas correctas de cada nodo el protocolo no intercambia mensajes ni tiene ninguna función. Cuando una ruta hacia un nuevo destino es necesaria, el nodo que la necesita envía una mensaje broadcast RREQ que llega al destino, o a un nodo intermedio que tiene una ruta suficientemente “fresca” hacia el destino. Una ruta es “fresca” cuando el número de secuencia hacia el destino es como mínimo tan grande como el número que contiene el RREQ. La ruta se considera disponible por el envió de un mensaje RREP hacia el nodo que originó el RREQ.

Los nodos monitorizan el estado de las conexiones de los nodos, a un salto, participantes en las rutas activas. Cuando una conexión se rompe en una ruta activa, se envía un mensaje RERR para notificar a los otros nodos la pérdida de la conexión.

Este protocolo tiene una tabla de rutas. La información de la tabla de rutas debe guardarse incluso para las rutas de corta vida. Los campos que tiene cada entrada de la ruta son los siguientes:

- IP de destino.
- Número de secuencia de destino.
- *Flag* número de secuencia de destino válido.
- Otros estados y *flags* de enrutamiento (válido, invalido, reparable...).
- Interfaz de red.
- Contador de saltos.
- Salto siguiente.
- Listado de precursores.
- Tiempo de vida.

1.2.1 Terminología

En este apartado se definen algunos nombres y sus significados que se utilizan en este protocolo:

- Ruta activa: una ruta que tiene una entrada en una tabla y esta marcada como válida. Sólo estas rutas se pueden usar para la retransmisión.
- Broadcast: estos paquetes no deben ser transmitidos por la red en exceso, pero son útiles para la transmisión de los mensajes del AODV por la red.
- Nodo retransmisión: nodo que permite la retransmisión de paquetes hacia otros nodos, por medio de enviar los paquetes hacia el siguiente salto.
- Ruta de retrasmisión: una ruta configurada para enviar paquetes de datos desde el nodo que origina el descubrimiento de la ruta hacia el destino deseado.
- Ruta inválida: una ruta que ha expirado, tiene el estado inválido. Estas rutas se utilizan para guardar una ruta válida anterior y de este modo tener la información durante más tiempo. Una ruta inválida no puede ser utilizada para la retransmisión de paquetes.
- Nodo originario: un nodo que inicia el mensaje de descubrimiento de ruta para ser procesado y poder ser retransmitido por otros nodos.
- Ruta contraria: una ruta configurada para retransmitir el paquete (RREP) desde el destinatario hacia el que ha originado el mensaje.
- Número de secuencia: un número incremental que mantiene cada nodo originario. En los mensajes del protocolo AODV se usa por los otros nodos para determinar la “frescura” de la información que tiene el nodo originador.

1.2.2 Mantenimiento de números de secuencia

Cada entrada de la tabla de cada nodo debe incluir la última información sobre el número de secuencia para la dirección IP del nodo destino. Este número de secuencia se llama “número de secuencia de destino”. Se actualiza cada vez que un nodo recibe nueva información del número de secuencia por los mensajes RREQ, RREP o RERR. Este protocolo depende de que cada nodo de la red mantenga su propio número de secuencia de destino para garantizar que no haya bucles. Un nodo destinatario incrementa su propio número de secuencia en dos circunstancias:

- Inmediatamente antes que un nodo origine el descubrimiento de una ruta, debe incrementar su propio número de secuencia.
- Inmediatamente antes que el nodo destino origine un mensaje RREP como respuesta a un RREQ, este nodo debe actualizar su número de secuencia, eligiendo el valor máximo entre su actual número de secuencia o el número del paquete RREQ que le ha llegado.

1.2.3 Entradas de la tabla de enrutamiento

Cuando un nodo recibe un paquete de control desde un vecino, crea o actualiza una ruta hacia un destino particular o una subred, el nodo comprueba su tabla de enrutamiento por una entrada para el destino. La ruta se actualiza en los siguientes casos:

- El número de secuencia es mayor que el que hay en al tabla de enrutamiento.
- El número de secuencia es igual, pero el nuevo valor del contador de saltos más uno, es menor que el valor que tenia la ruta de la tabla de enrutamiento.
- El número de secuencia es desconocido.

Las entradas de la tabla tiene un campo de tiempo de vida, este tiempo se determina por el paquete de control que llega, o se toma un valor determinado “ACTIVE_ROUTE_TIMEOUT”.

1.2.4 Generación de peticiones de rutas

Un nodo envía un mensaje RREQ cuando determina que necesita saber la ruta hacia un destino y no lo tiene en su tabla de enrutamiento o es una entrada no válida. En ese momento se envía un mensaje RREQ con el valor del número de secuencia de destino igual al último número conocido para este destino. El valor del número de secuencia de origen en el mensaje RREQ es el número de secuencia del nodo que es incrementado antes del envío del mensaje.

Al tener en cuenta que las comunicaciones son bidireccionales, además de la ruta para llegar al destino también es necesario saber una ruta de vuelta. Para este cometido cualquier nodo intermedio que genere un mensaje de respuesta

(RREP) debe también realizar una acción que notifique al nodo destino una ruta de vuelta hacia el nodo origen.

Para no crear congestión en la red ni hacer que los mensajes circulen indefinidamente por ella, el nodo que origina peticiones debe indicar un TTL máximo a los mensajes y además seleccionar un *timeout* para esperar una respuesta. Tanto el *timeout* como el TTL son calculados de manera periódica y tiene en cuenta el tamaño de la red y el tiempo que tarda un paquete en cruzarla.

1.2.5 Procesamiento y retransmisión de peticiones de ruta

Cuando un nodo recibe un RREQ, crea o actualiza una ruta hacia el salto anterior. Posteriormente comprueba que no haya recibido una mensaje con el mismo ID y origen y si lo ha recibido descarta este nuevo mensaje. En este apartado se explicará las acciones que se realizan cuando este mensaje no se descarta.

Lo primero que se hace es aumentar el valor del contador de saltos en uno. Después, el nodo busca una ruta hacia la IP origen del mensaje. Si no existe se debe crear esta nueva ruta de vuelta. Una vez se ha creado esta ruta de vuelta se siguen las siguientes acciones:

- El número de secuencia origen se compara con el número de secuencia hacia el destino que se tiene en la tabla, y si es mayor se copia en ella.
- Se valida el campo de número de secuencia.
- El siguiente salto en la tabla de enrutamiento se convierte el nodo desde donde nos ha llegado el mensaje.
- Se copia el número de saltos en la tabla de enrutamiento.

1.2.6 Generación de respuesta de ruta

Un nodo genera un mensaje RREP si él mismo es el destino, o tiene una ruta activa hacia el destino y el número de secuencia de la entrada de la tabla es mayor que el del mensaje RREQ. Una vez se genera el RREP el nodo descarta el mensaje RREQ.

Si un nodo no genera un RREP y el valor del TTL es mayor de uno entonces actualiza y envía el mensaje RREQ a una dirección *broadcast*.

Si el nodo que genera el mensaje RREP no es el nodo destino sino que es un nodo intermedio, copia su propio número de secuencia para el destino en el campo de número de secuencia destino del mensaje RREP. Entonces este nodo intermedio actualiza la ruta de retransmisión poniéndose a él como último nodo en la lista de precursores.

1.2.7 Recepción y retransmisión de respuesta de ruta

Cuando un nodo recibe un mensaje RREP busca una ruta hacia el salto anterior, si es necesario se crea esta ruta. Posteriormente el nodo incrementa el contador de saltos en el mensaje. Entonces se crea una ruta para llegar al destino si no existe. De otra manera, el nodo compara el número de secuencia de destino del mensaje con el que tiene guardado. Después de la comparación la ruta existente se actualiza en los siguientes casos:

- El número de secuencia en la tabla de enrutamiento está marcado como inválido.
- El número de secuencia de destino en el mensaje es mayor que el que el nodo tiene guardado y el valor es válido.
- Los números de secuencia son iguales pero la ruta está marcado como inactiva.
- Los números de secuencia son los mismos, y el nuevo valor del contador de saltos es menor.

Cuando se actualiza una entrada en la tabla la ruta se marca como activa, el número de secuencia de destino también se marca como válido y en el siguiente salto en la entrada de la tabla se asigna el nodo del que ha llegado el mensaje RREP. También se debe actualizar el nuevo valor del contador de saltos, el tiempo de expiración de la ruta y el número de secuencia de destino, se debe actualizar por el número de secuencia del mensaje RREP.

1.2.8 Mensajes de error (RERR)

Normalmente una ruta errónea o el corte de un enlace necesitan un procedimiento similar. Primero invalidar las rutas existentes, listar los destinos afectados, determinar los vecinos afectados y enviar un mensaje apropiado RERR a estos vecinos.

Un nodo inicia el procesamiento de un mensaje RERR en tres situaciones:

- Si detecta la caída de un enlace para el siguiente salto de una ruta activa en su tabla de enrutamiento mientras envía datos.
- Si recibe un paquete de datos hacia un nodo del que no tiene ninguna ruta activa.
- Si recibe un mensaje RERR desde un vecino por una o más rutas activas.

1.2.9 Ejemplo de utilización

En el dibujo posterior se puede ver como un nodo (A) busca la ruta hacia otro nodo (K) del que no conoce el camino. Lo primero que hace el nodo A es enviar un mensaje broadcast RREQ hacia todos los nodos, preguntando por el nodo J, con el que se quiere comunicar. Cuando el mensaje RREQ llega al nodo J este genera un mensaje RREP de respuesta. Este mensaje se envía como unicast de vuelta hacia el nodo A utilizando las entradas en memoria de los nodos H, G y D.

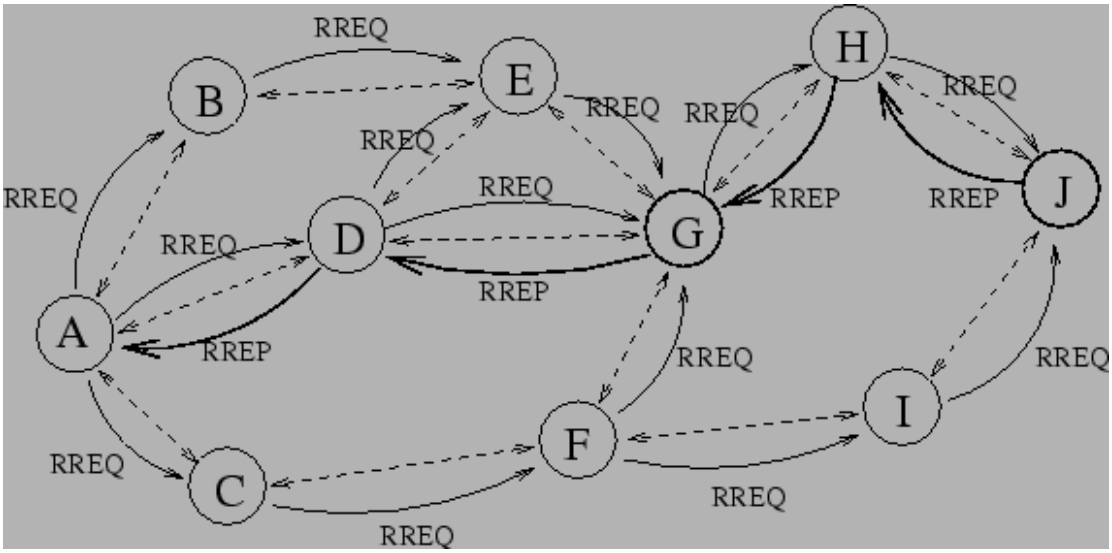


Fig 1.1 Ejemplo de búsqueda de un nuevo nodo

1.3 OSPF

El protocolo OSPF propone el uso de rutas más cortas y accesibles mediante la construcción de un mapa de la red y el mantenimiento de bases de datos con información sobre sistemas locales y vecinos. De esta manera es capaz de calcular la métrica para cada ruta, entonces se eligen las rutas de encaminamiento más cortas.

Todos los routers de OSPF tienen una base de datos detallada con la información necesaria para construir un árbol de encaminamiento del área, con la descripción de las interfaces, conexiones y métricas de los routers. Además de todas las redes de multiacceso y una lista de todos los routers de la red.

Los routers envían periódicamente mensajes de saludo (*Hello*), para que el resto de los routers sepan que siguen activos. También envían mensajes de saludo al otro extremo de un enlace punto a punto o un circuito virtual para que estos vecinos sepan que siguen atentos.

Una de las razones por las que funcionan los mensajes de saludo es que un mensaje contiene la lista de todos los identificadores de los saludos cuyos vecinos escucharan el emisor, así los routers conocen si se les está escuchando en la red.

1.3.1 Tipos de mensajes OSPF

Los cinco tipos de mensajes del protocolo OSPF que se han utilizan son:

- Saludo: Se usa para identificar a los vecinos.
- Descripción de la base de datos: Durante la inicialización, se usa para intercambiar información de manera que un router puede descubrir los datos que le faltan en la base de datos.
- Petición del estado del enlace**:** Se usa para pedir datos que un router se ha dado cuenta que le faltan en su base de datos o que están obsoletos.
- Actualización del estado del enlace**:** Se usa como respuesta a los mensajes de Petición del estado del enlace y también para informar dinámicamente de los cambios en la topología de la red.
- ACK de estado del enlace: Se usa para confirmar la recepción de una Actualización del estado del enlace. El emisor retransmitirá hasta que se confirme.

1.3.2 Router designado

En una red multiacceso, los mensajes de saludo también se usan para identificar a un *Router designado*. El router designado cumple dos funciones:

Es responsable de la actualización fiable de sus vecinos adyacentes con la información más reciente de la topología de la red.
Crea avisos de enlaces de red con la lista de todos los routers conectados a la red multiacceso.

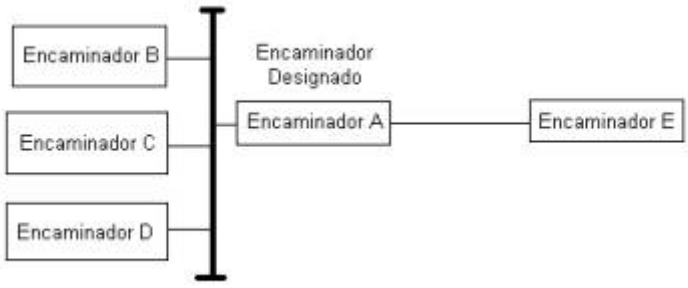


Fig 1-2 Dibujo de router designado

El router designado A intercambia información con los routers B, C y D de su LAN, así con el router E conectado con su enlace punto a punto.

1.3.3 Adyacencias

El router designado A actúa como experto local y mantiene actualizada la topología local completa. Después comunica a los routers adyacentes la información. B, C y D mantienen sus propias bases de datos sincronizadas hablando con A. No tienen que hablar con los otros, así se reduce drásticamente el tráfico de información. Dos routers que sincronizan sus bases de datos uno con otro se llaman adyacentes. B y C son vecinos, pero no son adyacentes el uno del otro debido a que consultan con A.

1.3.4 Sistemas autónomos de área

Un área es un conjunto de redes y host contiguos, junto con routers con interfaces a estas redes. Un sistema autónomo que use OSPF está construido por una o más áreas. Cada área tiene asignado un número. El área 0 está conectada al Backbone que enlaza con el resto de áreas y agrupa al resto de sistemas autónomos.

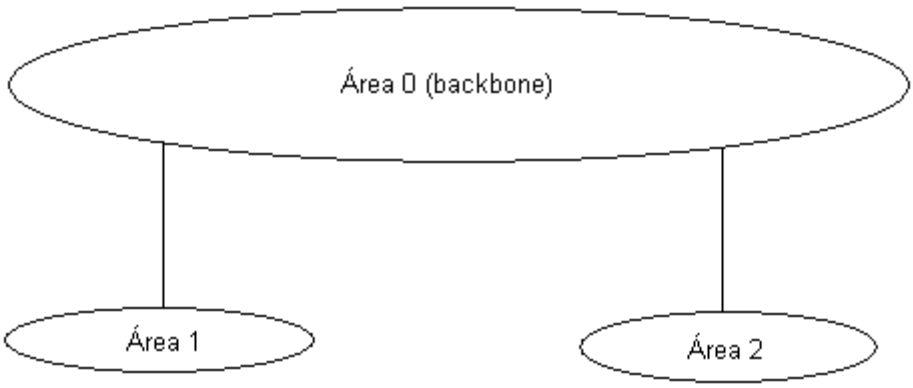


Fig 1-3 Backbone y áreas en OSPF

La red agrupa las áreas. El backbone contiene todos los routers que pertenecen a múltiples áreas, así como las redes y routers no asignados a ninguna área. Se debe recordar que las áreas están numeradas y que el backbone es el área 0. Un router *frontera* pertenece a una o más áreas y al backbone.

1.3.5 Encaminamiento de área en OSPF

El encaminamiento dentro de un área se basa en un mapa completo de estado de enlace del área. Todos los routers con OSPF implementado en un área mantienen una base de datos de encaminamiento idéntica que describe la topología y estado de todos los nodos de esa área. La base de datos se usa para construir el mapa de esa área. Siempre que ocurre un cambio, la información se propaga por toda el área. De esta forma siempre los routers

estarán en un estado óptimo para cualquier petición. Un router que esté arrancando obtendrá una copia de la base de datos actual de encaminamiento de su vecino más cercano (vecino se denomina a cualquier router que esté en su área).

El router de frontera conoce la topología completa de las áreas a las que esta conectado. Estos routers resumen la información de área e indican a otros routers del backbone lo lejos que están de las redes dentro de su propia área. De esta forma todos los routers frontera pueden calcular las distancias a destinos fuera de sus propias áreas y transmitir esta información dentro de sus propias áreas.

1.3.6 Destino fuera de los Sistemas Autónomos de OSPF

Muchos sistemas autónomos están conectados a Internet, o a otros sistemas autónomos. Los enrutadores límite de OSPF ofrecen información sobre distancias a las redes externas al sistema autónomo.

Existen dos tipos de métricas de distancia externa de OSPF. La de tipo 1 es equivalente a la métrica local de estado del enlace. Las métricas de tipo 2 de larga distancia, se miden con un mayor orden de magnitud.

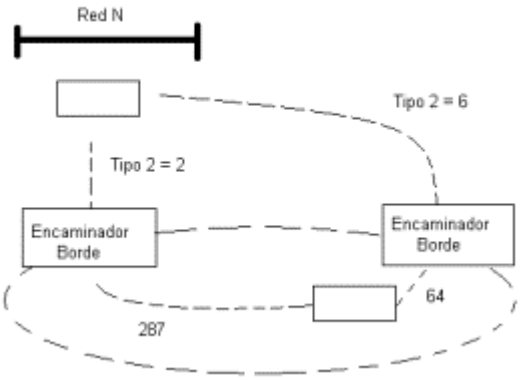


Fig 1-4 Dibujo de elección de métrica

En el dibujo vemos que existen dos rutas, hemos elegido la métrica de tipo 2, para llegar a la red externa vemos que eligiendo la de tipo 2 la mas corta es la de valor 2. Otra característica de OSPF conveniente para los Proveedores de Servicios Internet es que un router límite de un sistema autónomo puede comportarse como un *servidor de encaminamiento* y puede informar de las entradas que identifican las rutas a otros routers límite.

1.3.7 Wireless

El protocolo OSPF tiene un gran funcionamiento para redes cableadas pero tiene distintos problemas en las redes sin cables como la aleatoriedad y los

cambios continuos de la infraestructura. Para solucionar estos problemas se desarrollan versiones posteriores que incluyan funcionalidades de enrutamiento MANET (Mobile Ad-hoc NETwork).

En este sentido se han desarrollado distintos draft intentando dar una solución a los distintos problemas que tienen las redes sin cables. Aún no existe ningún estándar en este sentido y se esta trabajando en ellos desde distintas perspectivas:

- Problem Statement for OSPF Extensions for Mobile Ad Hoc Routing. En las redes ad hoc muchas de las características y capacidades de OSPF son necesarias. En este documento trata de explicar los cambios necesarios que se deberían hacer para que este protocolo funcionara correctamente en redes de este tipo.
- Design Considerations for a Wireless OSPF Interface. En este draft se presentan analisis y resultados de simulaciones de un diseño del protocolo OSPF con interfaces inalámbricas.
- OSPFv2 Wireless Interface Type. Describe mejoras al protcolo OSPFv2 para el soporte de interfaces inalámbricas, con capacidad broadcast y para redes multi-hop.
- Extensions to OSPF to Support Mobile Ad Hoc Networking. Este documento especifica un mecanismo para la señalización local de enlaces, una interfaz OSPF-MANET, un método para reducir el tamaño de los paquetes Hello y un método para optimizar la inundación de actualizaciones de enrutamiento.
- MANET Extensión of OSPF using CDS Flooding. Describe y evalua una familia de métodos de inundación para las redes ad hoc.

Las principales problemas que suponen este tipo de redes al protocolo OSPF son la siguientes:

- Mobilidad y cambios de la geografía en la que se encuentran los nodos (routers) puede llevar a muchas situaciones complejas con vecinos que tienen cobertura por dos routers distintos, o dificultan la elección del mejor camino hacia el destino.
- Adaptación de la red al entorno gráfico. Posibilidad de usar antenas direccionales o no.
- Minimización del tráfico retransmitido. En OSPF hay optimizaciones de inundación de mensajes como el uso de DR, pero estas optimizaciones son basándose en redes con multiacceso. Sin embargo, las redes inalámbricas muchas veces no cumplen esta característica, y no se puede asumir que la conectividad es en ambos sentidos. Por este motivo estas optimizaciones no son óptimas si no se combinan con otras como la inundación del estado de los enlaces (LSF). Se debe utilizar un mecanismo que permita sincronizar la información de todos los nodos del segmento.
- Limitar transmisión. Se debe minimizar también el tráfico originado. Una solución estudiada es la de que los nodos envíen un mensaje LSA y asuman que se encuentran en la base de datos LSA hasta que no sea explícitamente retirado.

- Control de potencia. Al ser dispositivos móviles que pueden actuar con baterias es muy importante el uso que se le da. Los dispositivos que utilicen este tipo de energía sólo actuaran y retransmitiran mensajes siempre que sea necesario porque ningún otro vecino pueda encargarse.
- Gran número de vecinos inmediatos. En una interfaz inalámbricas puede haber un gran número de vecinos adyacentes, además en este tipo de redes puede haber un gran número de nodos con características de routers. Por estos dos motivos los protocolos que trabajen en este tipo de redes deben poder dar soporte a un gran número de adyacencias y permitir fácilmente la llegada de nuevos vecinos.
- Movilidad rápida. La topología de la este tipo puede cambiar bruscamente y un gran número de veces en poco tiempo cuando la red es muy densa, estos cambios tan variados no pueden llevar a un gran número de mensajes que colapsen la red para la creación de adyacencias o la caída de enlaces. Por este motivo deben haber cambios en el establecimiento de vecinos, resumen de topologías y inundación de mensajes LSA.
- Inundación fiable o no. Para la inundación de mensajes LSA se puede utilizar transmisión multicast pero el reconocimiento llegará desde cada vecino. Esto hace que aumente considerablemente el tráfico, se ha trabajado con distintas soluciones para minimizar este tráfico como TBRPF (Topology Dissemination Based on Reverse-Path Forwarding) que tiene routers que transmiten los LSAs hacia sus vecinos necesarios para que estos puedan calcular las routas hacia ellos mismos, asumiendo que otros mensajes LSA llegarán hacia ellos por otras rutas si realmente son necesarios.

1.4 OLSR (Optimized Link State Routing Protocol)

Este protocolo desarrollado para redes móviles ad hoc, opera en modo proactivo. Cada nodo selecciona un grupo de nodos vecinos como “multipoint relay” (MPR), en este caso sólo los nodos seleccionados como tales son responsables de la retransmisión de tráfico de control. Estos nodos también tienen la responsabilidad de declarar el estado del enlace a los nodos que los tienen seleccionados como MPR.

Es muy útil para redes móviles densas y grandes, porque la optimización que se consigue con la selección de los MPR trabaja bien en estos casos. Cuanto más grande y densa sea una red mejor es la optimización que se consigue con este protocolo. OLSR utiliza un enrutamiento salto-a-salto, es decir, cada nodo utiliza su información local para enlutar los paquetes.

La selección de los nodos MPR reduce el número de retransmisiones necesarias para enviar un mensaje a todos los nodos de la red. OLSR optimiza la reacción a cambios en la topología reduciendo el intervalo de transmisión de los mensajes periódicos de control. Como este protocolo mantiene rutas hacia

todos los destinos de la red trabaja muy bien en redes donde el tráfico es aleatorio y esporádico entre un gran número de nodos.

OLSR trabaja de manera distribuida sin ninguna entidad central. Este protocolo no requiere transmisiones seguras de mensajes de control porque los mensajes son periódicos, y se pueden permitir algunas pérdidas. Tampoco necesita una recepción de mensajes secuencial, se utiliza números de secuencia incrementales para que el receptor sepa que información es más reciente.

1.4.1 Terminología

Palabras claves para entender el funcionamiento de este protocolo:

- Nodo: Router que implementa el protocolo OLSR.
- Interfaz OLSR: interfaz de un equipo que participa en el protocolo OLSR. También se debe tener en cuenta que hay otras interfaces en estos equipos que no trabajan en el protocolo.
- Dirección principal: la dirección principal de un nodo, se utilizará como la dirección de origen del tráfico de control en OLSR emitida por este nodo.
- Nodo vecino: un nodo X es vecino de otro nodo Y, si el nodo Y puede escuchar nodo X. Existe un enlace entre los dos nodos. Dos nodos son vecinos si ambos se encuentran dentro del área de cobertura del otro.
- Vecino a 2 saltos: un nodo “escuchado” por un vecino. Nodo, no vecino, que está dentro del área de cobertura de un nodo vecino.
- Vecino a 2 saltos estricto: un nodo que es vecino de un vecino del nodo que se esta mirando.
- MPR (Multiponit relay): un nodo que es seleccionado por su vecino, nodo X, para retransmitir todos los mensajes *broadcast* que recibe del nodo X.
- MPR selector (MS): un nodo que ha seleccionado su vecino como su MPR. MPS de un nodo x es todo aquel que tiene a x como MPR
- Enlace: pareja de interfaces OLSR sensibles a “escuchar” el otro. Los enlaces pueden ser simétricos (enlace bidireccional), asimétricos (sólo verificados en un sentido).
- Vecindario simétrico de 1 salto: de un nodo X es el grupo de nodos que tiene un enlace simétrico hacia X.

1.4.2 Funcionamiento

OLSR está modulado para tener un núcleo de funcionalidades, que siempre es requerido, y un grupo de funcionalidades auxiliares.

1.4.2.1 Funcionamiento núcleo

El núcleo especifica el comportamiento de un nodo que tiene interfaces OLSR. Se basa en las siguientes funcionalidades:

- Formato de paquete y retransmisión.

OLSR se comunica mediante un formato de paquete unificado para todos los datos del protocolo. El propósito de esto es facilitar la extensión del protocolo. Estos paquetes se envían como datagramas UDP.

Cuando recibimos un paquete básico, un nodo examina el mensaje, y basándose en un campo donde se indica el tipo de mensaje determinará el procesamiento del mensaje que seguirá los siguientes pasos:

- Si el paquete no contiene mensaje (el tamaño es demasiado pequeño) se descarta.
- Si el valor del TTL es menor o igual que 0 también se descarta.
- Condiciones de proceso. Si es un mensaje es duplicado (la dirección de origen y la número de secuencia ya se han tratado) no se procesa. En caso contrario el paquete es tratado de acuerdo al tipo de mensaje que haya llegado.
- Condiciones de retransmisión. Si es un mensaje duplicado no se retransmite, si no es duplicado se retransmite el mensaje siguiendo el algoritmo del tipo de mensaje.

- Percepción de enlace

Se consigue saber el estado del enlace mediante el envío de mensajes “HELLO”. El propósito de esta funcionalidad es que cada nodo tenga asociado un estado en el enlace a cada uno de sus vecinos. El estado puede ser simétrico (enlace verificado es bidireccional) y asimétrico indica que los mensajes “HELLO” se han escuchado pero no podemos asegurar que este nodo escuche las respuestas.

- Detección de vecino

Dada una red de nodos con sólo una interfaz, un nodo debe deducir los vecinos que tiene mediante la información intercambiada durante la percepción de enlace. Cada nodo debe tener guardados su grupo de vecinos. Cada vecino debe tener asociado el estado del enlace.

Cuando se detecta la aparición de un nuevo enlace, se debe crear una entrada con un vecino que tiene un enlace asociado, en esta entrada también se debe guardar el estado de este enlace. Se debe tener en cuenta que cada vez que varía el estado del enlace se debe comprobar en la tabla que el cambio se lleva a cabo. Si no se recibe información de un enlace durante un tiempo establecido se debe borrar el enlace en cuestión y el vecino asociado.

- Selección de MPR y señalización MPR. La selección de los MPR sirve para seleccionar los nodos vecinos que se quiere que hagan *broadcast* de los mensajes de control. La señalización viene dada mediante mensajes “HELLO”. Cada nodo elige uno o más MPRs de manera que

se asegura que a través de los MPRs seleccionados, cada nodo llega a todos los vecinos a dos saltos.

- Difusión de mensajes de control de topología. Estos mensajes se difunden con el objetivo de dar a cada nodo de la red la información necesaria para permitir el cálculo de rutas, son llamados mensajes TC (Topology Control). Estos mensajes que retransmite un nodo hacia sus vecinos seleccionados como MPR, tienen la información de todos sus enlaces para que los otros nodos conozcan los vecinos a los que puede llegar.

- Cálculo de rutas. Dada la información del estado del enlace que se adquiere mediante el intercambio de mensajes periódicos. Cada nodo mantiene una tabla de enrutamiento que permite encaminar los paquetes de datos destinados a otros nodos. Esta tabla esta basada en la información contenida en las bases de información de enlace y de la topología. Esta tabla se actualiza cuando se detecta algún cambio en estos campos:
    - El enlace
    - El vecino
    - El vecino de dos saltos
    - La topología

1.4.2.2 Funciones auxiliares

Hay situaciones donde funcionalidades auxiliares son necesarias, como por ejemplo un nodo con múltiples interfaces, donde algunas de ellas participan en el otro dominio de enrutamiento.

- Interfaces no OLSR. Hay nodos que pueden tener interfaces que no son OLSR, estas interfaces pueden ser conexiones punto a punto o conectar con otras redes. Para poder tener conectividad entre las interfaces OLSR y estas otras el router debe ser capaz de introducir información externa de encaminamiento a la red. Para esto las interfaces no OLSR crean un mensaje Host and Network Association (HNA) que contiene información suficiente para poder crear nuevas rutas con esta información.

- Notificación capa enlace. OLSR no trabaja con información de capa enlace. Sin embargo, si la información de esta capa está disponible, esta información se utiliza además de la información de los mensajes “HELLO”, para mantener información de los vecinos y los MPR. Por ejemplo: la pérdida de conectividad de la capa de enlace se puede deber a la ausencia de reconocimientos de capa de enlace.

- Información redundante de topología. Para poder proveer redundancia a la información de topología, la información de anuncio que emite el nodo ha de tener información de enlaces hacia nodos vecinos que no

necesariamente tengan a este nodo como MPR. El mensaje de anuncio publica información de todos los enlaces de los nodos vecinos. Hay tres posibles niveles de redundancia:

- Sin redundancia: sólo se emite información del grupo que ha elegido a este nodo como MPR.
- Redundancia media: se emite información del grupo que ha elegido el nodo como MPR y también información de los nodos que este ha elegido como MPR.
- Redundancia alta: se emite información de todos los enlaces hacia los vecinos.

- MPR redundante. Esta funcionalidad especifica la habilidad del nodo de seleccionar MPR redundantes. Aunque la redundancia crea mucho más tráfico y pierde eficiencia el mecanismo de MPR, se tiene una gran ganancia al asegurar la llegada de los paquetes a sus destinos. Esta funcionalidad es útil para situaciones en que la red tiene mucha movilidad y mantener una buena cobertura con los MPR.

1.4.3 Ejemplo de utilización

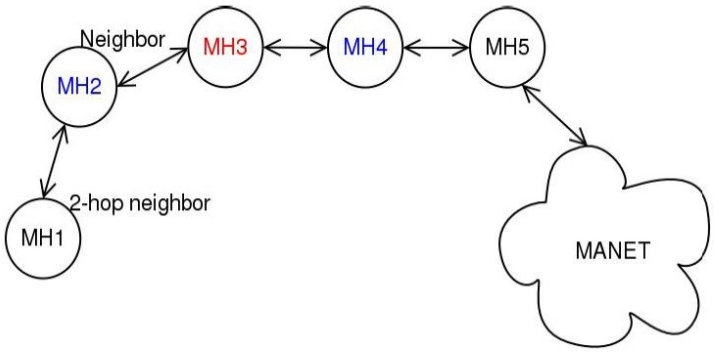


Fig 1.5 Topología de la red

En este dibujo podemos ver una red con 5 nodos colocados de manera estratégica para que todos ellos tengan un vecino a cada lado. En la siguiente tabla podemos ver un ejemplo de la tabla de enrutamiento del nodo MH3.

		1	2	3
MH2	Asim. Link		MH3	
	Sim. Link	MH1	MH1	MH1,MH3
	2-hop neighbours			MH4
MH3	Asim. Link		MH2,MH4	
	Sim. Link			MH2,MH4
	2-hop neighbours			MH1,MH5
MH4	Asim. Link		MH3	
	Sim. Link	MH5	MH5	MH3,MH5
	2-hop neighbours			MH2

Tabla 1 Tabla de enrutamiento del nodo MH3

En la tabla podemos ver que los vecinos pueden estar en dos estados como enlace asimétrico o simétrico según la calidad de los enlaces en el momento en el que llegan los paquetes. Cada una de las columnas de la tabla indican un momento del proceso de recepción de paquetes de señalización. En la tercera columna se puede observar que ya ha llegado a converger la red. En cambio en las dos primera columnas había nodos que no se habían detectado o incluso algunos que se habían detectado pero no se había comprobado la comunicación en ambos sentidos. También se puede ver en esta tabla como los vecinos a dos saltos son aquellos que son vecinos de algún nodo que tenemos en el estado de enlace simétrico.

Capítulo 2. Creación de la maqueta

En este apartado se pretende dar una visión del proceso de instalación, configuración y utilización de la red. Esta red consta de puntos de acceso inteligentes que son los que se conectan a otra red y permiten a los demás clientes conectarse a ellos. Esta red utilizará IPv6 por este motivo lo primero que se debe realizar es la instalación de un sistema operativo que permita trabajar con este protocolo. El nodo inteligente consiste en un ordenador personal equipado con dos tarjetas de red inalámbrica. Los ordenadores de esta red están equipados con dos tarjetas inalámbricas pcmcia. Una de estas tarjetas, basada en el chipset Prism 2, está controlada por un *driver* llamado Hostap[2] que permite utilizar un ordenador como si fuera un AP. Para las redes configuradas en este proyecto se han utilizado 4 ordenadores que serán configurados como nodos de la red. En la siguiente tabla se pueden ver las características principales de estos nodos, así como los drivers utilizados para sus interfaces Inalámbricas, son 4 ordenadores 2 PCs de sobremesa más dos portátiles de la marca ACER, los dos portátiles tienen exactamente la misma configuración ya que son el mismo modelo y tienen las mismas instalaciones, en cambio los PCs de sobremesa utilizan interfaces distintas.

Característica	Portátiles	Sobremesa 1	Sobremesa 2
CPU	Intel Pentium M 1.5GHz	Pentium 2 350 MHz	Pentium 2 350 MHz
RAM	512MB DDR SDRAM	256 MB SDR	256 MB SDR
Sistema operativo	Fedora Core 2 2.6.9	Red hat kernel 2.4.24	Red hat kernel 2.4.24
Tarjeta Ad-hoc	Intel PRO/Wireless 2100	Cisco AIRONET 350	Cisco AIRONET 350
Driver	ipw2100-1.0.5	airo.c 0.2	airo.c 0.2
Tarjeta AP	Allied Telesyn AT-WCL452	Netgear MA401	CONCEPTRONIC CON11C
Driver	hostapd versión 0.3.7	hostapd versión 0.2.4	hostapd versión 0.2.4

Tabla 2 Características ordenadores utilizados

2.1 Instalación de IPv6

El primer paso consiste en cargar el módulo de IPv6 si el kernel está preparado para esta opción. Esta opción se activa con el siguiente comando:

```
modprobe ipv6
Can´t locate
```

El mensaje obtenido nos indica que en el kernel que se tiene instalado en el ordenador no es posible esta opción.

[2] http://hostap.epitest.fi/

Para poder usar ipv6 se deberá compilar de nuevo un kernel. En este caso tenemos un kernel 2.4.24. Para poder añadir el modulo IPv6 lo primero que debemos hacer es habilitar la opción antes de la recompilación. Lo que se pretende es simplemente añadir la funcionalidad de IPv6, por esto se debe buscar en la máquina el archivo de configuración del antiguo kernel (.config), una vez se encuentra y se observa que se tienen las fuentes se pasa a añadir la funcionalidad.

Para añadir la funcionalidad se puede activar desde el menú gráfico de configuración del kernel, accesible mediante el siguiente comando: *make xconfig,* y una vez ahí, tras activar las opciones experimentales (Code maturity level options/Prompt for development and/or incomplete code/drivers), se debe seleccionar la opción de IPv6 (networking options/IPv6).

Después de esto cabe recordar que es extremadamente importante que las opciones de pcmcia y de WLAN que estaban añadidas en la anterior compilación del kernel estén habilitadas. Una vez todo esté preparado, se debe recompilar el kernel con los siguientes comandos, dentro de la carpeta con las fuentes (para más información ver Linux Kernel Howto)[3]:

```
make dep
make bzImage
make modules
make modules_install
cp arch/i386/bzImage /boot/vmlinuz-2.4.24
```

2.2 Definición de la red

La red que se desea crear consta de dos tipos de nodos. Los nodos clientes y los nodos AP. Los nodos clientes, son simplemente los usuarios que quieren acceder a la red o a Internet mediante nuestro sistema. En cambio, los nodos AP son los encargados de dar acceso a la red a los clientes y también la señalización entre los distintos nodos APs de la red.

Nodo AP

Un nodo de este tipo debe contar con dos interfaces inalámbricas, una la encargada de dar acceso a los clientes, debe estar configurada en modo infraestructura, y la otra, que será la que se encargará de enviar mensajes de señalización estará en modo ad-hoc para que los APs vecinos tengan la información necesaria que se trata en esta red. En el siguiente dibujo se puede ver un ejemplo gráfico.

Un problema que se puede encontrar en el momento de usar las dos tarjetas al mismo tiempo es que por defecto el driver sólo utiliza una interrupción IRQ (Interrupt ReQuest), por este motivo se debe modificar en el script

[3] www.**linux**docs.org/HOWTOs/Kernel-HOWTO.html

/etc/config.opts que utiliza para que permita el uso de un par de interrupciones diferentes. Una vez se tienen las dos tarjetas instaladas y listas para utilizar se puede proceder a la configuración de la red.

Al querer utilizar más interrupciones, se debe realizar una reserva previa añadiendo las interrupciones que se deseen en el archivo de configuración /etc/config.opts. En este caso se reservan las interrupciones 3 y 4, por lo que añaden las siguientes líneas:

```
reserve irq 4
reserve irq 3
```

Antes de reservarlas se debe cerciorar de que no habrá conflicto con una irq usada por otro dispositivo; para ello se debe mirar las que están siendo utilizadas mediante el comando:

```
more /proc/interrupts
               CPU0
  0:           272497         XT-PIC  timer
  1:           4238           XT-PIC  keyboard
  2:           0              XT-PIC  cascade
  5:           11246          XT-PIC  es1371
  9:           0              XT-PIC  usb-uhci
 10:           10155          XT-PIC  eth0
 12:           28106          XT-PIC  PS/2 Mouse
 14:           13496          XT-PIC  ide0
 15:           3              XT-PIC  ide1
NMI:           0
ERR:           0
```

En este ejemplo se puede observar como no se están utilizando ni la 3 ni la 4, por este motivo se pueden reservar sin ningún problema. Una vez se conecta la primera interfaz se puede observar como la interrupción 3 es utilizada por el dispositivo:

```
more /proc/interrupts
               CPU0
  0:           282374         XT-PIC  timer
  1:           4772           XT-PIC  keyboard
  2:           0              XT-PIC  cascade
  3:           1998           XT-PIC  eth1
  5:           11246          XT-PIC  es1371
  9:           0              XT-PIC  usb-uhci
 10:           10464          XT-PIC  eth0
 12:           31342          XT-PIC  PS/2 Mouse
 14:           13611          XT-PIC  ide0
 15:           3              XT-PIC  ide1
NMI:           0
ERR:           0
```

Se debe tener en cuenta que en las redes inalámbricas se puede elegir canal en el que se quiere trabajar, se debe tener en cuenta que debe haber una separación entre canales, para esto hay que elegir el canal de las dos

interfaces, para que no haya interferencias, esta selección se hace mediante un script.

Las interferencias se producen por que se utiliza una banda libre, en la que no se paga para poder usar una frecuencia (2.412-2.472 para IEEE 802.11b/g), y como consecuencia se debe compartir la banda con otros usuarios. Esta banda esta dividida en 13 canales con una separación de 5 MHz. El ancho de banda de la señal es de unos 20MHz, por lo que, para que no se produzcan interferencias entre transmisiones debe haber una separación de 5 canales, es decir, si se detecta una nodo trabajando en el canal 3 para no tener interferencias deberíamos trabajar como mínimo en el canal 8. Por este motivo se deben elegir de manera metódica y precisa los canales en que se quiere trabajar para no tener interferencias ni desaprovechar la banda.

Este script lo primero que hace es la selección del canal ad-hoc. Primero escanea el espectro tratando de comprobar si hay un nodo vecino que esté trabajando en la red con la que se quiere trabajar, esta red tiene como ESSID “wmn”. Si se encuentra un nodo se debe elegir trabajar como ad-hoc, con el ESSID “wmn” y en el canal que está trabajando el otro nodo, si en caso contrario, no hay ningún nodo trabajando en esta red entonces se busca un canal vacío y que no tenga interferencias con co-canales.

Una vez elegido el canal ad-hoc, se pasa a trabajar en busca del canal para la interfaz infraestructura. En este caso sólo se debe buscar un canal vacío y en el que no haya excesivas interferencias. En la siguiente figura se puede observar el diagrama de funcionamiento.

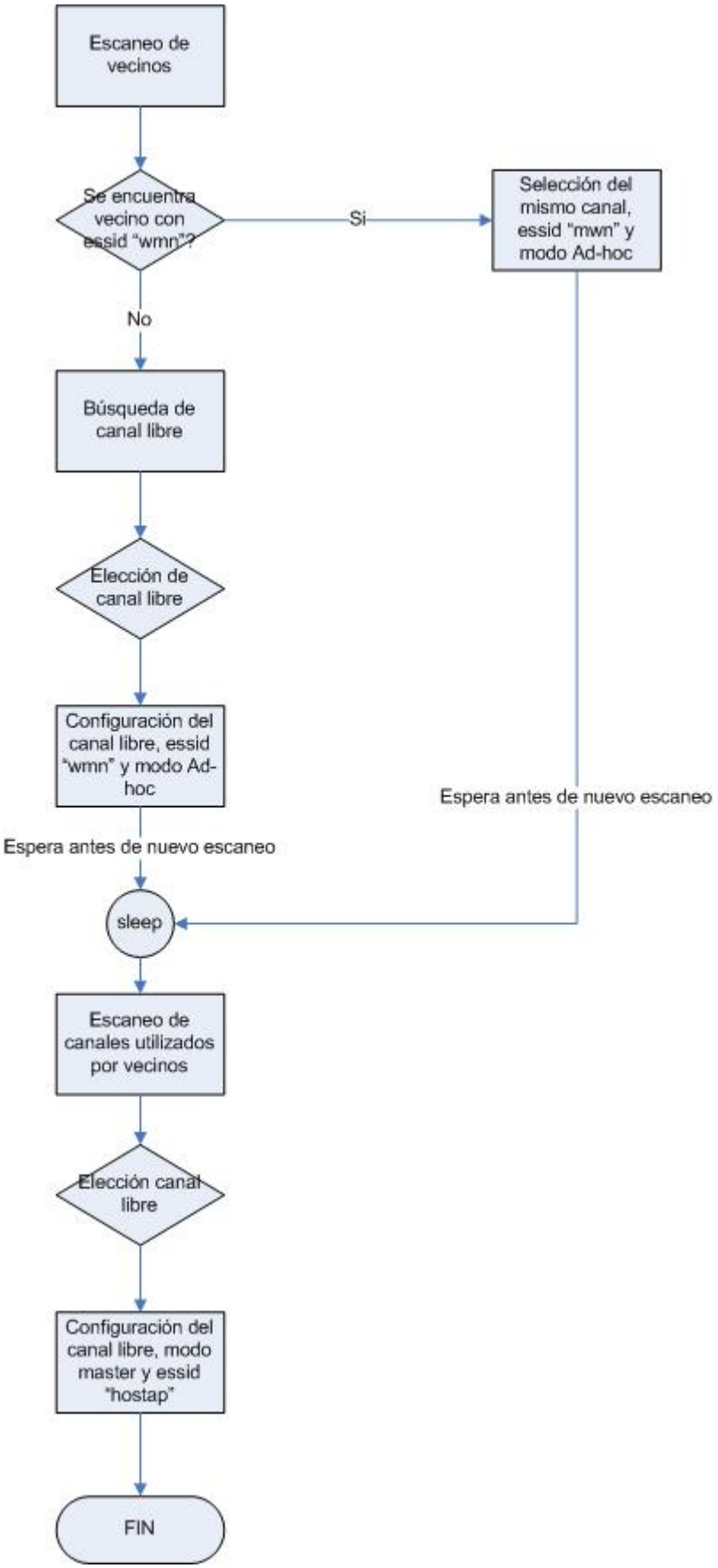


Fig 2.1 Diagrama de funcionamiento

Una vez se tiene las interfaces inalámbricas configuradas, los diferentes usuarios se pueden conectar a ellas mediante la red de infraestructura con el ESSID “hostap”. Se debe tener en cuenta que las interfaces “wmn” deben tener unas IPs de la misma subred, en este caso como se quiere que todas las interfaces de esta misma red puedan tener comunicación se les debe dar unas direcciones IPv6, estas IPs deben ser del mismo enlace para que los mensajes multicast de señalización lleguen a todos los nodos vecinos.

Las IPs de las interfaces de la red wmn como se ha dicho deben ser de la misma subred en la siguiente tabla se puede ver la correspondencia del ordenador físico con el nodo al que se refiere, también se indica la configuración de IPs que se le da a la red:

	Sobremesa 1	Portátil 1	Portátil 2	Sobremesa 2
Tipo IPv6	Nodo 1	Nodo 2	Nodo 3	Nodo MPR
Site	FEDE::A:1/112	FEDE::A:4/112	FEDE::A:3/112	FEDE::A:2/112
Link	FE80::A:1/112	FE80::A:4/112	FE80::A:3/112	FE80::A:2/112

Tabla 3 Configuración de los nodos de la red

En la siguiente figura se puede ver la configuración física de la red:

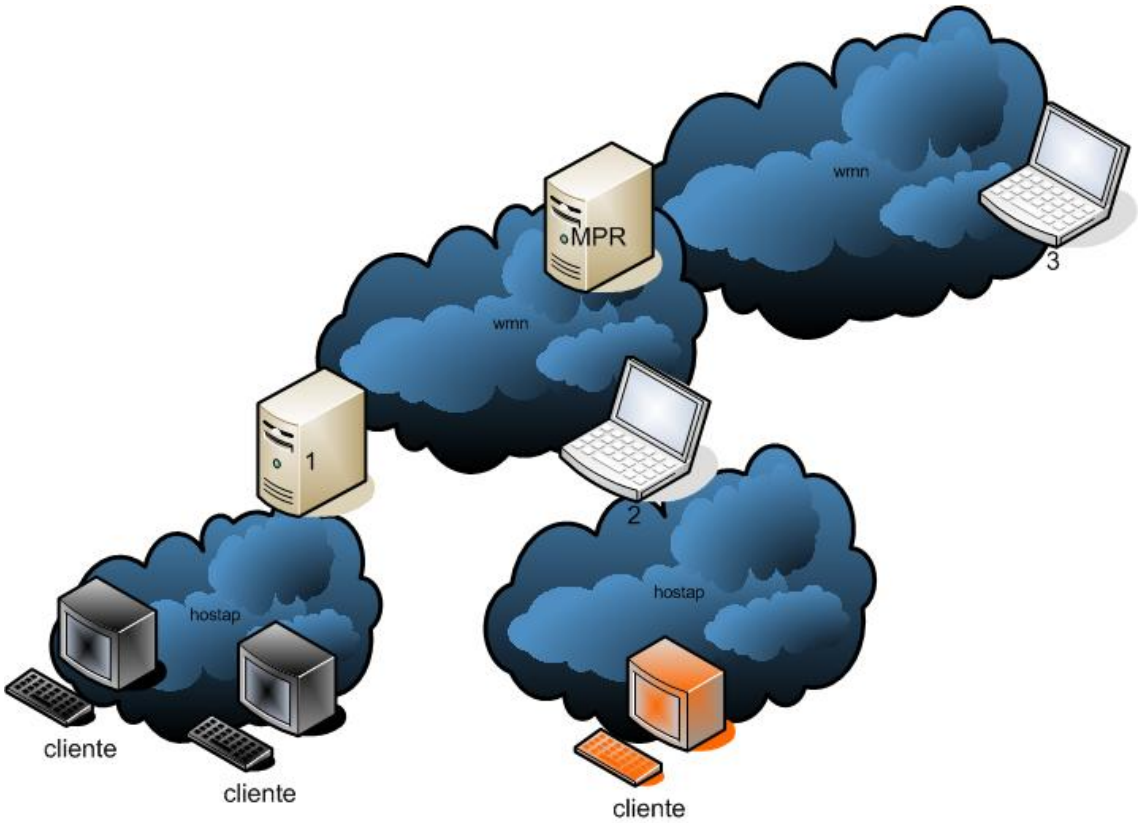


Fig 2.2 Esquema de una red

La conectividad entre los nodos se puede comprobar utilizando un ping6 hacia alguno de los nodos o un ping6 multicast hacia todos. Se debe tener en cuenta que sólo se podrán comunicar entre ellos si tienen visibilidad directa y están a una distancia prudencial. Para poder comunicar nodos que no tengan

visibilidad directa se deberá utilizar un protocolo de enrutamiento como los explicados en un apartado anterior.

Como se explicará más adelante, por las características del algoritmo de selección de frecuencias, se requiere el uso de un protocolo de encaminamiento tipo link state. Debido a no haber una implementación pública de OSPF para redes ad-hoc, en este caso se ha procedido a la utilización del protocolo OLSR, para poder tener comunicación entre todos los nodos de la red.

2.3 Instalación OLSR

La instalación de este protocolo empieza con la descarga del software necesario, para ello se descarga una de las múltiples implementaciones, la más conocida y probada es la versión de la universidad noruega UNIK[4]. La versión instalada de este software es la 0.4.9. Una vez descargadas las fuentes se debe instalar siguiendo los siguientes pasos:

- Descomprimir las fuentes
- `make OS=unix`
- `make install OS=unix`

Después de esto, ya se tiene instalado el protocolo, a partir de aquí se deben modificar ciertas opciones del archivo de configuración que se encuentra en /etc/olsrd.conf.

En este archivo se debe cambiar un par de opciones, la primera es la de selección de la versión del protocolo IP que se quiere utilizar, se debe elegir IPv6. Otra opción que se debe seleccionar es el tipo de IP que se utilizará para la publicación de los nodos (site, local o global), para la red que se tiene con las IPs site se puede trabajar sin dificultad.

Una vez se tiene el protocolo preparado se puede empezar a trabajar con él para ejecutarlo se debe elegir la interfaz con que trabajará el protocolo para ello el comando que se debe escribir es el siguiente: `olsrd –i eth1`.

2.4 Red ejemplo

Una vez se tiene el protocolo instalado se procederá a la creación de una maqueta en la que se probará la utilidad y funcionamiento de éste. La red que se pretende crear tiene un nodo que sólo tiene visibilidad directa con uno de los nodos en cambio, no puede ver a los demás, gracias al protocolo de enrutamiento podrá llegar a todos los nodos de la red.

[4] www.olsr.org

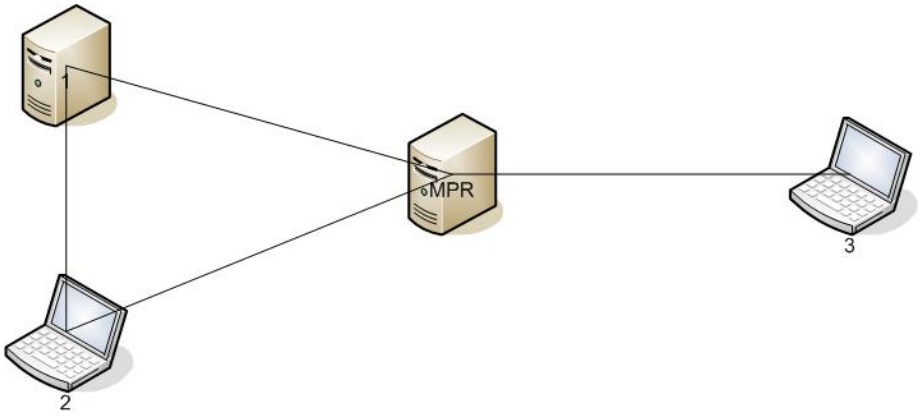


Fig 2.3 Dibujo de la maqueta

Como se observa en el dibujo los nodos 1 y 2 tienen visibilidad directa entre ellos y además, con el nodo MPR. Este nodo MPR también tiene visibilidad directa con el nodo 3.

Para conseguir esta red no es necesario alejar el nodo 3 de modo que no vea los demás nodos, lo que se ha hecho es crear listas de acceso en las que algunos nodos se excluían. Para esto se debe utilizar ip6tables, ya que lo que se pretende es trabajar con IPv6. El comando utilizado para esta funcionalidad es:

```
ip6tables –A INPUT –m mac –mac-source xx:xx:xx:xx:xx:xx –j
DROP
```

Este comando se debe ejecutar en el nodo 3 con las MACs de los nodos 1 y 2. Una vez hecho esto y comprobado su funcionamiento mediante ping6 se debe ejecutar el demonio OLSR en los 4 nodos indicando la interfaz correcta, la interfaz que se quiere usar es la que se usa como ad-hoc.

Una vez se tiene todo correctamente configurado y se ejecuta el protocolo en los 4 nodos se procede al intercambio de paquetes. Como se tiene configurada la red el nodo que se ha llamado MPR será el seleccionado como MPR. En la siguiente figura se puede observar como un nodo lo selecciona como MPR y es mostrado por pantalla. En este caso se puede observar como el nodo con dirección FEDE::a:6 ha seleccionado ha este nodo como MPR por eso es añadido a MPRS.

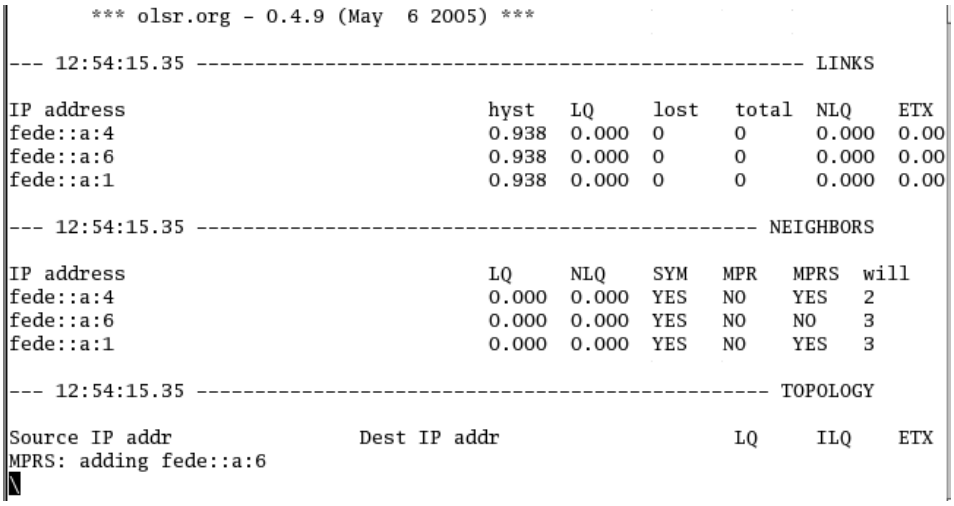


Fig 2.4 Selección de nodo MPR

Los demás nodos seleccionan al nodo como MPR, en su tabla se puede observar como han elegido al nodo con dirección FEDE::A:3 como nodo MPR y además también se puede observar como sólo se consigue que el enlace sea simétrico con los nodos con visibilidad directa. En la siguiente figura se puede ver los dos nodos.

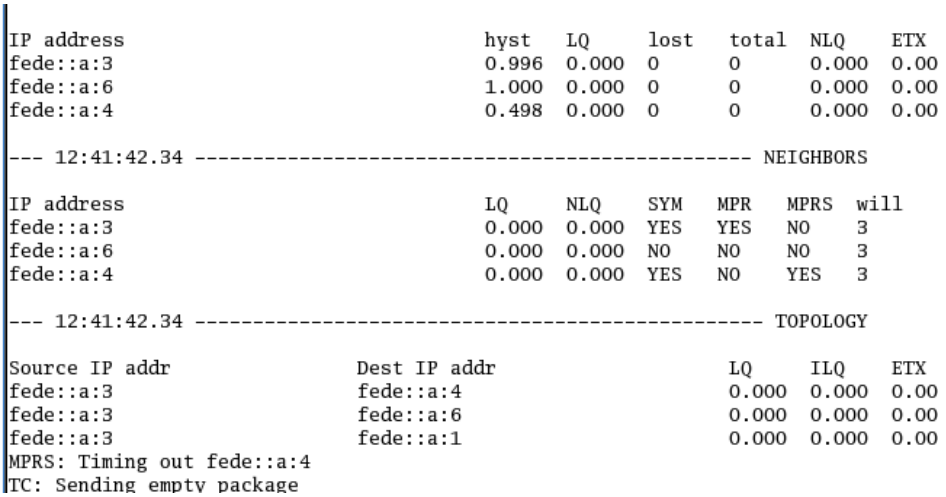


Fig 2.5 Nodo 1

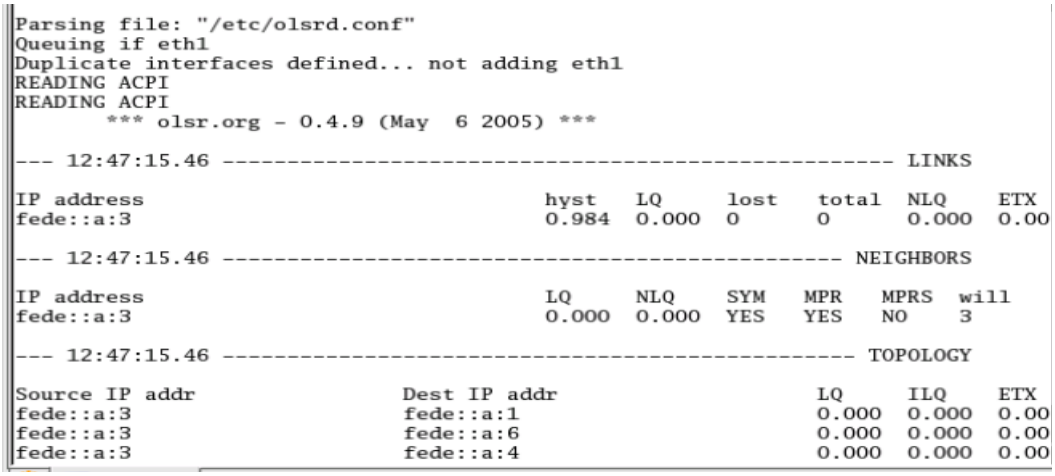


Fig 2.6 Nodo 3

En la figuras se puede observar como el nodo 3 sólo tiene un vecino con visibilidad directa que es el que selecciona como MPR.

2.5 Red ejemplo avanzada

Una vez se tiene configurado el protocolo de enrutamiento se quiere avanzar en la construcción de la maqueta. Debido a que en este caso simplemente se tiene una red Ad-hoc con un protocolo de enrutamiento. Lo que se desea es utilizar las prestaciones que se han comentado en el apartado anterior (selección de canal, uso de sistemas de aviso y demás) en esta red. Para la selección de canal se utilizará un algoritmo basado en la coloración de grafos. Para que este algoritmo pueda ser ejecutado en todos los nodos, es indispensable que todos los nodos tengan información de los nodos vecinos, o incluso los nodos de los que no se recibe señal pero que un nodo intermedio tal vez si reciba. Es decir, todos los nodos deben conocer las interferencias que reciben los demás nodos.

Para ello es indispensable que todos los nodos tengan información fundamental de los nodos más cercanos no sólo de su MPR. Para que todos los nodos tengan información se debe configurar el protocolo de enrutamiento para que trabaje con redundancia.

Para conseguir trabajar con redundancia se deberá modificar el archivo de configuración del protocolo de enrutamiento que se encuentra en /etc/olsrd.conf. Dentro del archivo se encuentra el comando para selección de redundancia, **TcRedundancy**. Este comando permite tres tipos de redundancia (0, 1 y 2). Como se explico en el apartado teórico de OLSR, esta redundancia se puede dar en tres tipos, sin redundancia (0), redundancia de la información de los MPR y MPRS (1) y por último redundancia de MPR y los vecinos de este nodo (2). Para este proyecto será necesaria la mayor redundancia posible (la tercera opción). Se debe mencionar que por defecto la redundancia es mínima (0). En el script se podría leer así:

```
TcRedundancy    2
```

Una vez actualizado todos los archivos de configuración es posible comprobar las diferencias respecto al proceso anterior (sin redundancia). La red con la que se pretende trabajar es la misma con un nodo que no tiene acceso a los otros dos, por lo que la selección de MPR será la misma.

Una vez configurados todos los nodos de la red, con todas las características descritas en los párrafos anteriores. En las siguientes figuras se puede comprobar las tablas de enrutamiento que obtienen los nodos.

```
IP address                              hyst   LQ     lost   total  NLQ    ETX
fede::a:4                               0.906  0.000  0      0      0.000  0.00
fede::a:2                               1.000  0.000  0      0      0.000  0.00
fede::a:3                               1.000  0.000  0      0      0.000  0.00

--- 15:54:46.78 ------------------------------------------------- NEIGHBORS

IP address                              LQ     NLQ    SYM    MPR    MPRS   will
fede::a:4                               0.000  0.000  YES    NO     NO     3
fede::a:2                               0.000  0.000  YES    YES    YES    3
fede::a:3                               0.000  0.000  NO     NO     NO     3

--- 15:54:46.78 ------------------------------------------------- TOPOLOGY

Source IP addr                Dest IP addr                      LQ     ILQ    ETX
fede::a:3                     fede::a:2                         0.000  0.000  0.00
fede::a:2                     fede::a:3                         0.000  0.000  0.00
fede::a:2                     fede::a:4                         0.000  0.000  0.00
fede::a:2                     fede::a:1                         0.000  0.000  0.00
fede::a:4                     fede::a:2                         0.000  0.000  0.00
fede::a:4                     fede::a:1                         0.000  0.000  0.00
```

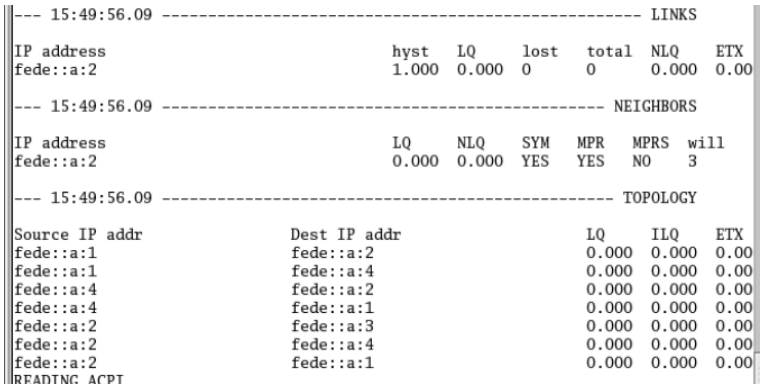
Fig 2.7 Tabla enrutamiento nodo 1

```
--- 15:47:59.80 ------------------------------------------------- LINKS

IP address                              hyst   LQ     lost   total  NLQ    ETX
fede::a:1                               1.000  0.000  0      0      0.000  0.00
fede::a:2                               1.000  0.000  0      0      0.000  0.00
fede::a:3                               1.000  0.000  0      0      0.000  0.00

--- 15:47:59.80 ------------------------------------------------- NEIGHBORS

IP address                              LQ     NLQ    SYM    MPR    MPRS   will
fede::a:1                               0.000  0.000  YES    NO     NO     3
fede::a:2                               0.000  0.000  YES    YES    NO     3
fede::a:3                               0.000  0.000  NO     NO     NO     3

--- 15:47:59.80 ------------------------------------------------- TOPOLOGY

Source IP addr                Dest IP addr                      LQ     ILQ    ETX
fede::a:1                     fede::a:2                         0.000  0.000  0.00
fede::a:1                     fede::a:4                         0.000  0.000  0.00
fede::a:3                     fede::a:2                         0.000  0.000  0.00
fede::a:2                     fede::a:3                         0.000  0.000  0.00
fede::a:2                     fede::a:4                         0.000  0.000  0.00
fede::a:2                     fede::a:1                         0.000  0.000  0.00
```

Fig 2.8 Tabla enrutamiento nodo 2

```
--- 15:57:05.65 ------------------------------------------------- LINKS

IP address                              hyst   LQ     lost   total  NLQ    ETX
fede::a:1                               1.000  0.000  0      0      0.000  0.00
fede::a:4                               1.000  0.000  0      0      0.000  0.00
fede::a:3                               0.902  0.000  0      0      0.000  0.00

--- 15:57:05.65 ------------------------------------------------- NEIGHBORS

IP address                              LQ     NLQ    SYM    MPR    MPRS   will
fede::a:1                               0.000  0.000  YES    NO     YES    3
fede::a:4                               0.000  0.000  YES    NO     YES    3
fede::a:3                               0.000  0.000  YES    NO     YES    3

--- 15:57:05.65 ------------------------------------------------- TOPOLOGY

Source IP addr                Dest IP addr                      LQ     ILQ    ETX
fede::a:1                     fede::a:2                         0.000  0.000  0.00
fede::a:1                     fede::a:4                         0.000  0.000  0.00
fede::a:4                     fede::a:2                         0.000  0.000  0.00
fede::a:4                     fede::a:1                         0.000  0.000  0.00
fede::a:3                     fede::a:2                         0.000  0.000  0.00
```

Fig 2.9 Tabla de enrutamiento nodo MPR



```
--- 15:49:56.09 ------------------------------------------------- LINKS

IP address                              hyst   LQ     lost   total  NLQ    ETX
fede::a:2                               1.000  0.000  0      0      0.000  0.00

--- 15:49:56.09 ---------------------------------------------- NEIGHBORS

IP address                              LQ     NLQ    SYM   MPR   MPRS  will
fede::a:2                               0.000  0.000  YES   YES   NO    3

--- 15:49:56.09 ---------------------------------------------- TOPOLOGY

Source IP addr               Dest IP addr                  LQ     ILQ    ETX
fede::a:1                    fede::a:2                     0.000  0.000  0.00
fede::a:1                    fede::a:4                     0.000  0.000  0.00
fede::a:4                    fede::a:2                     0.000  0.000  0.00
fede::a:4                    fede::a:1                     0.000  0.000  0.00
fede::a:2                    fede::a:3                     0.000  0.000  0.00
fede::a:2                    fede::a:4                     0.000  0.000  0.00
fede::a:2                    fede::a:1                     0.000  0.000  0.00
READING ACPI
```

Fig 2.10 Tabla enrutamiento nodo 3

Se puede comprobar que en este caso la tabla de enlaces y de vecinos, así como la selección de MPR es la misma que sin redundancia. La diferencia viene dada en la tabla donde se indica la topología, en ella se puede ver que indica como llegar a todos los nodos. Ahora se nos presentan todos los nodos vecinos y a dos saltos, indicando cuáles son los nodos que tienen un enlace hacia él, no sólo nos dan la información de los nodos vecinos del MPR, sino que nos da la información de estos nodos que se añade a la tabla de topología. De este modo se tiene la información completa de la topología conociendo los enlaces no sólo que unen con el MPR sino que además enlaces desconocidos por este. Por ejemplo, el enlace entre 1 y 2, antes no era conocido por 3 porque al no haber redundancia el MPR simplemente notifica los enlaces que eran conocidos por él, es decir, sólo notifica el enlace que tenia con los nodos a los que estaba conectado mediante un enlace y no los enlaces de estos nodos.

2.5.1 Intercambio de mensajes

En este apartado se puede ver el funcionamiento práctico del protocolo OLSR. Se quiere ver los mensajes que se producen durante el lanzamiento del servicio hasta la convergencia.

En este ejemplo hay dos nodos que están trabajando en el mismo canal, uno de estos nodos tiene el servicio olsrd arrancado por lo que envía mensajes de este protocolo. Al cabo de unos segundos se arranca en otro nodo el servicio olsrd. Mediante el software ethereal (versión 0.10.11) se recibirán los mensajes hasta que se llegue a una situación de convergencia entre los dos nodos, la situación de convergencia se entiende como el momento en que los dos nodos se conocen y el enlace se considera simétrico. En la siguiente figura se puede observar el diagrama de secuencia. El destino de todos los mensajes es ff05::15, una dirección multicast para que pueda llegar a todos los nodos de con OLSR de la red.

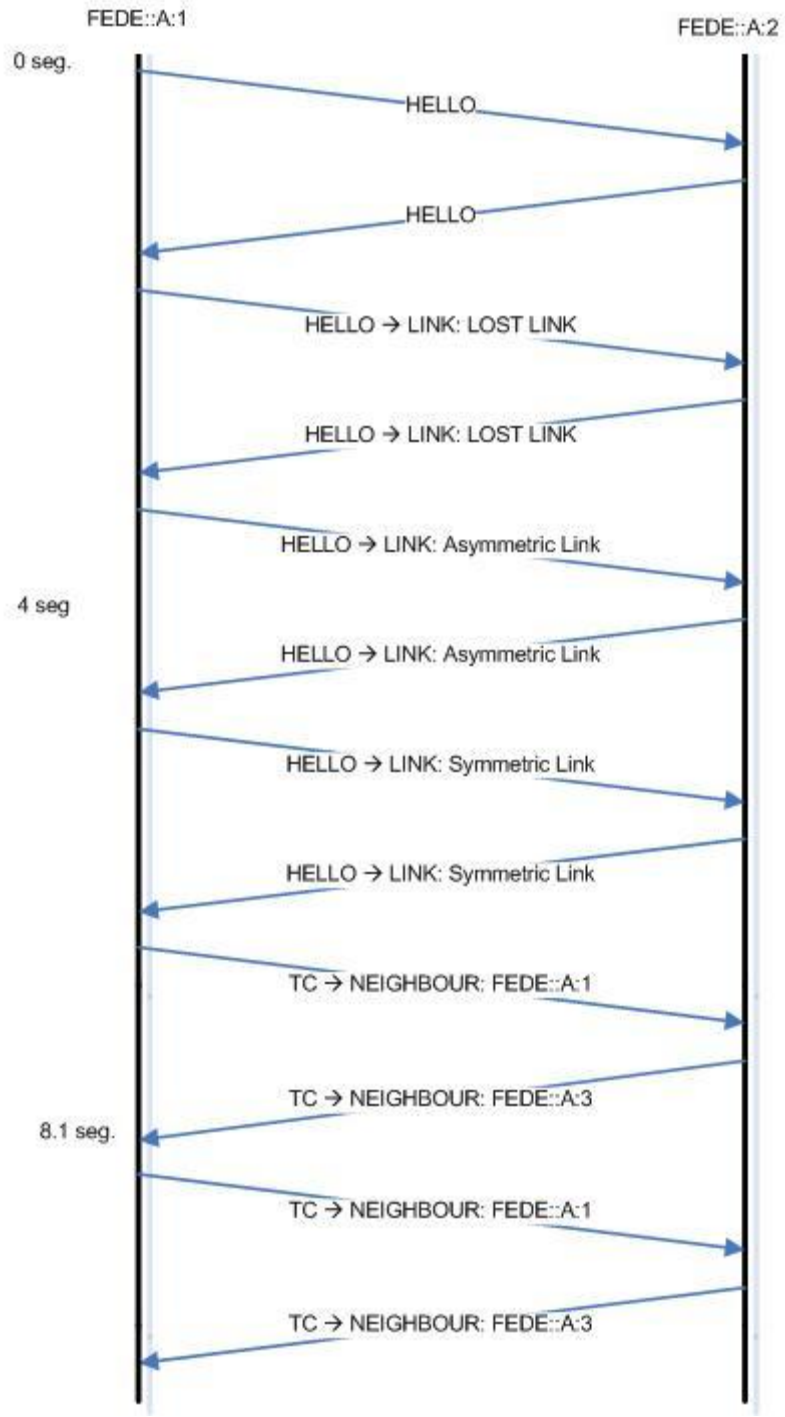


Fig 2.11 Diagrama de secuencia

En este diagrama además de poder ver el flujo de mensajes OLSR entre dos enlaces distintos, se puede observar el tiempo de convergencia, es decir, el tiempo que tarda un nuevo nodo en ser detectado y configurado correctamente dentro de las tablas de enrutamiento de los nodos vecinos. En este ejemplo el tiempo ha sido de 8.1 segundos hasta que se ha recibido el mensaje TC indicando el nuevo vecino.

El mensaje HELLO tiene distintas funcionalidades, la primera es la de encontrar nuevos vecinos en el radio de alcance. Una vez detecta un nodo detecta un vecino, recibe un HELLO del otro, empieza la detección del enlace, empieza

indicando que el enlace está perdido si sólo ha recibido un HELLO, si posteriormente recibe otro mensaje indicando que el otro nodo también lo detecta (mensaje con enlace pérdido), los siguientes mensajes son para detectar si el enlace es simétrico o asimétrico. Si un nodo envía HELLO indicando asimetría y el otro nodo le responde del mismo modo es que el enlace es simétrico, se reconoce porque en el mensaje se indica la dirección del vecino. A partir de ese momento se envían los mensajes indicando que el enlace es simétrico.

En la siguiente figura se puede ver el formato del mensaje HELLO:

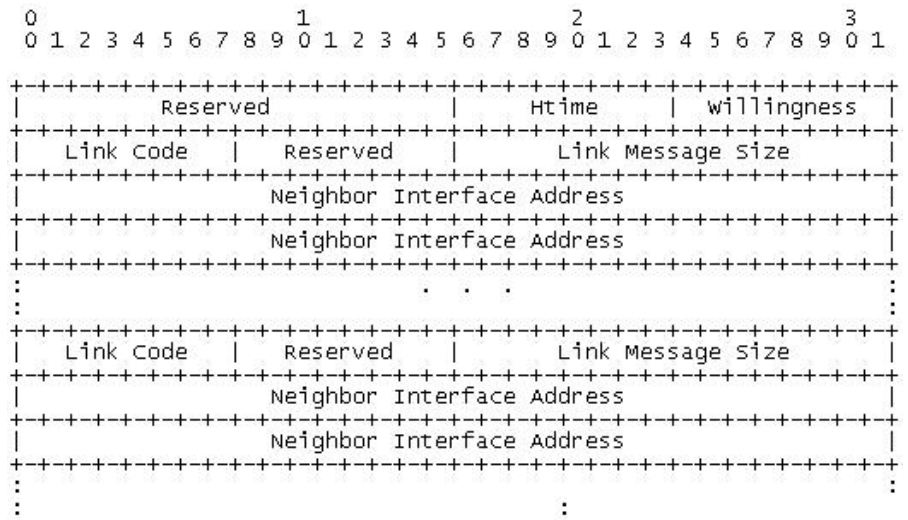


Fig 2.12 Formato del mensaje HELLO

En el formato del mensaje hay distintas informaciones, cuando no se ha detectado ningún vecino algunas de ellas están vacías, pero una vez se detecta un vecino se debe indicar el código del enlace (si es asimétrico, simétrico, perdido), así como el vecino con el que se esta negociando el enlace.

El mensaje TC tiene la siguiente estructura, también se debe comentar que si alguna interfaz tiene que enviar un mensaje HELLO y uno TC en el mismo instante en lugar de enviar dos paquetes en un mismo paquete envía los dos mensajes.

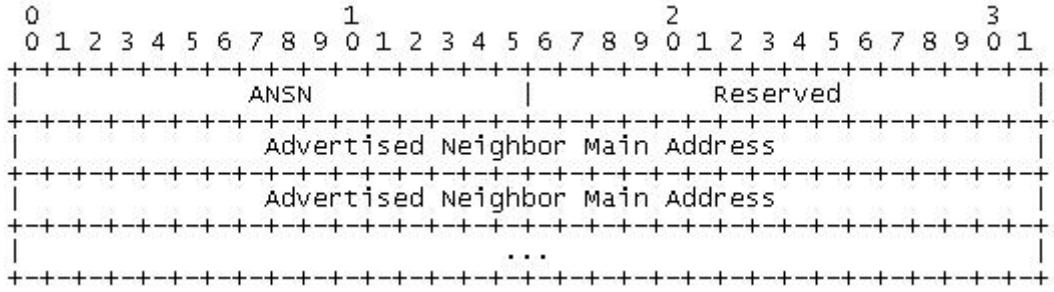


Fig 2.13 Mensaje TC

Este paquete se retransmite cada 15 segundos, mientras que el paquete HELLO es cada 2 segundos. En este mensaje se envía el número de

secuencia del mensaje, pero la información es otra, primero se indica quien es el nodo que ha originado el mensaje y posteriormente da la lista de las direcciones de sus vecinos, en este caso sólo uno.

2.5.2 Selección de MPR

Una vez conocidos todos los vecinos y indicando que el estado del enlace es simétrico se procede a la selección del MPR, esta selección la realiza el nodo y para indicarla lo hace mediante mensajes HELLO. Una vez ha elegido el MPR en su mensaje HELLO en el lugar donde indica el estado del enlace se debe poner MPR. En la siguiente figura se tiene una captura de pantalla de un mensaje que ha sido recibido por un nodo mediante el software Ethereal:

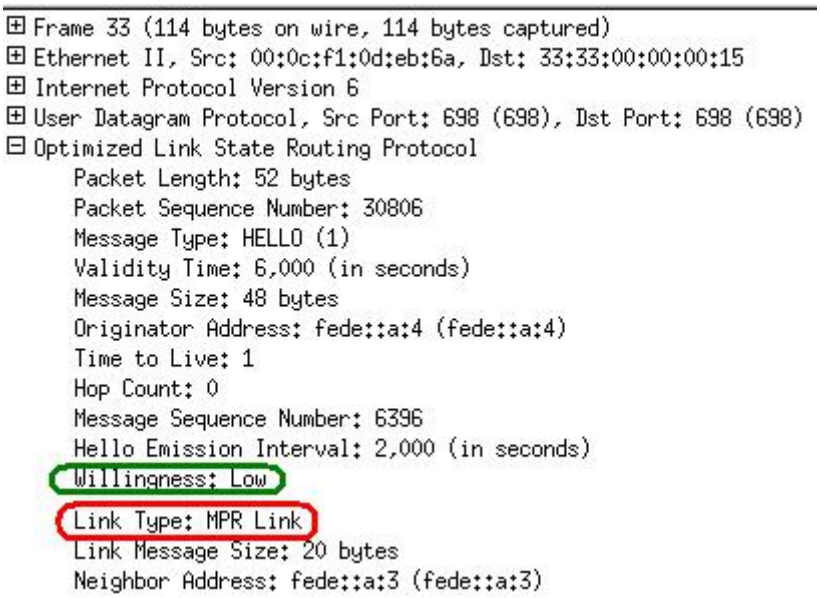


Fig 2.14 Mensaje HELLO indica MPR

Este nodo corresponde a un portátil, es por este motivo que el valor willingness es bajo (low). Esto es debido a que como este ordenador utiliza batería y no está enchufado a la red eléctrica, pues se debe elegir un valor bajo para evitar en la medida que sea posible ser elegido como MPR. De este modo no gasta tanta energía. Se debe recordar que el valor willingness indica la prioridad del nodo para ser elegido como MPR, cuando menor sea este valor menor probabilidad.

Para comprobar el correcto funcionamiento y la conectividad entre todos los nodos de la red se ha procedido a la ejecución de pings entre todos los nodos de esta.

Capítulo 3. Implementación de un protocolo de intercambio de información

Como las redes móviles ad-hoc (MANET) son un área aún en desarrollo, la distribución UNIK del protocolo de encaminamiento OLSR, que es la utilizada para este proyecto tiene en cuenta este problema, y por este motivo permite añadir nuevas funcionalidades mediante plugins. Esto permite añadir nuevos mensajes permitiendo que el protocolo siga en todo caso las especificaciones de los RFCs.

Esta funcionalidad será utilizada en este proyecto para el envío de mensajes privados. Los mensajes enviarán información acerca de los nodos que crean interferencias a este nodo, con la información del canal que utilizan los vecinos que crean estas interferencias y la potencia recibida de ellos. Toda esta información se guardará para saber en cada momento que interferencias tienen todos los nodos e incluso las interferencias que tienen los nodos vecinos.

3.1 Añadiendo funcionalidades mediante plugins

El proceso olsrd permite la carga de librerías dinámicas (DLL), llamadas plugins. Estos son usados para la generación y procesado de mensajes privados y cualquier otra funcionalidad. Una DLL es una parte de código que contiene funciones y datos. En este caso se usarán como plugins, es decir, darán nuevas funciones al protocolo existente sin alterar su funcionamiento normal. Se debe tener en cuenta que en Linux las DLLs son conocidas como *.so*. En la siguiente figura se puede observar el uso de estos DLL.

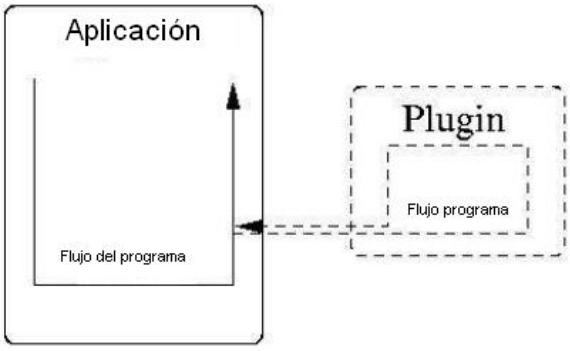


Fig 3.1 Ejemplo interacción de la aplicación y el programa

Los plugins se pueden escribir en cualquier lenguaje que se pueda compilar como una librería dinámica. Las nuevas versiones de olsrd podrán utilizar los mismos plugins sin necesidad de cambios.

OLSR permite la retransmisión de mensajes OLSR aunque sean tipos desconocidos. Esto quiere decir que aunque sólo algunos de los nodos sepan como interpretar estos mensajes, estos serán retransmitidos por todos los

nodos de la red. Además también permite cambiar las funcionalidades y tratamientos que se le dan a los mensajes conocidos. Por ejemplo mediante un plugin se puede cambiar el tratamiento que se da cuando llega un mensaje HELLO. Esto implica que mediante un plugin se puede manipular cualquier parte del demonio olsrd que se tiene en la máquina.

Para esta manipulación es necesaria una interfaz que comunique perfectamente el demonio olsrd y los plugins. Con esta interfaz el olsrd sabe siempre qué debe recibir de los plugin y al contrario. El olsrd se puede dividir en distintas partes:

- Socket parser: comprueba el tráfico que llega al nodo y posteriormente llama a la función asociada con los sockets que tienen los datos que han llegado.
- Packet parser: recibe todo el tráfico OLSR que se recibe. Tiene tres responsabilidades básicas. Descartar el mensaje si es inválido. Enviar el mensaje hacia la función adecuada. Reenviar el mensaje de acuerdo con el algoritmo de retransmisión, esto se realiza si el mensaje es válido pero no se tiene ninguna función asociada a este tipo de mensaje.
- Depósitos de información. Las tablas de información. Dónde se guarda la información fresca de todos los cálculos y mensajes recibidos.
- Planificador de eventos. Se encarga de realizar las distintas funciones a diferentes intervalos. Si se quiere retransmitir un mensaje en un intervalo se debe registrar la creación de un paquete en el planificador.

En la siguiente figura se puede observar cómo el plugin tiene comunicación con todas las partes que forman el demonio OLSR.

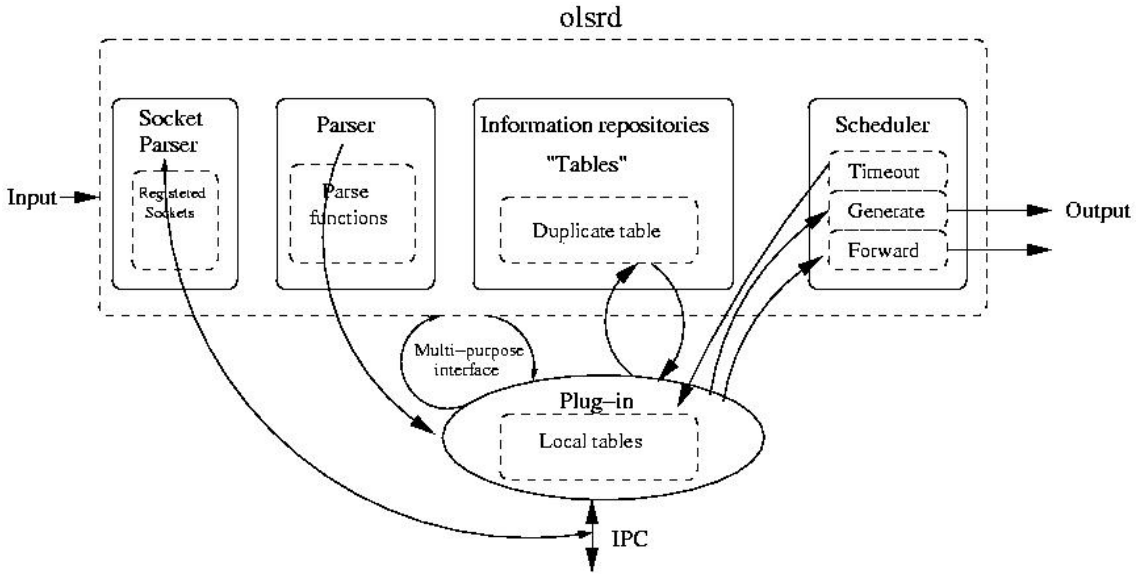


Fig 3.2 Dibujo de la interfaz entre olsrd y plugin

3.2 Protocolo de intercambio de información

Este es el plugin desarrollado para el proyecto. En él se trata de añadir un nuevo mensaje al protocolo de modo que siga con su funcionamiento normal. Además estos mensajes deben ser retransmitidos entre todos los nodos de la red para poder adquirir información de las interferencias que reciben todos los nodos.

El plugin se ha desarrollado con el lenguaje C siguiendo el manual[5] de la página oficial de UNIK OLSR. Lo primero que se debe hacer es crear una estructura donde se indicará qué información se enviará en el mensaje. Se debe tener en cuenta que el mensaje tendrá las mismas características que los mensajes habituales, es decir, tipo, tiempo de validez, etc. Entonces mediante una función de la interfaz se introduce un nuevo tipo de mensaje (en este caso el tipo 128), con esto se fuerza que el olsrd cuando recibe un mensaje de este tipo lo envíe a una función del plugin que será la encargada de tratarla y añadirla a nuestra tabla. Esta tabla no se mostrará por pantalla como las otras tablas de enrutamiento sino que para ser mostrada se deberá conectar mediante telnet al puerto 8888, en ese momento mostrará toda la información que reciba ese nodo. Se debe tener en cuenta que como se quiere que estos mensajes se retransmitan a un cierto intervalo se debe registrar la función de generación de mensajes en el planificador.

El funcionamiento del plugin es sencillo, a la función encargada de seleccionar el tipo de mensaje e indicar con qué función debe ser tratado, se le debe añadir un nuevo tipo que enviará el mensaje a una función que deberá tratar el mensaje como se desee. Para tratarlo se debe tener en cuenta el formato del mensaje y de la información que se va a enviar. Lo que en este caso se quiere transmitir es la MAC del nodo, el canal y la utilización de esta interfaz, es decir el tráfico que circula por ella. Además de la información del propio nodo también se debe dar a conocer los valores que se reciben de los vecinos del nodo. Esta información es similar a la del propio nodo y consta de MAC del nodo vecino, el canal en el que trabaja, y la potencia recibida de este nodo vecino. Además de todos estos datos después de los valores del nodo se deberá enviar un número indicando la cantidad de vecinos que se reciben. En la siguiente figura se puede ver fácilmente la información que contiene el mensaje.

[5] http://www.olsr.org/docs/olsrd-plugin-howto.html

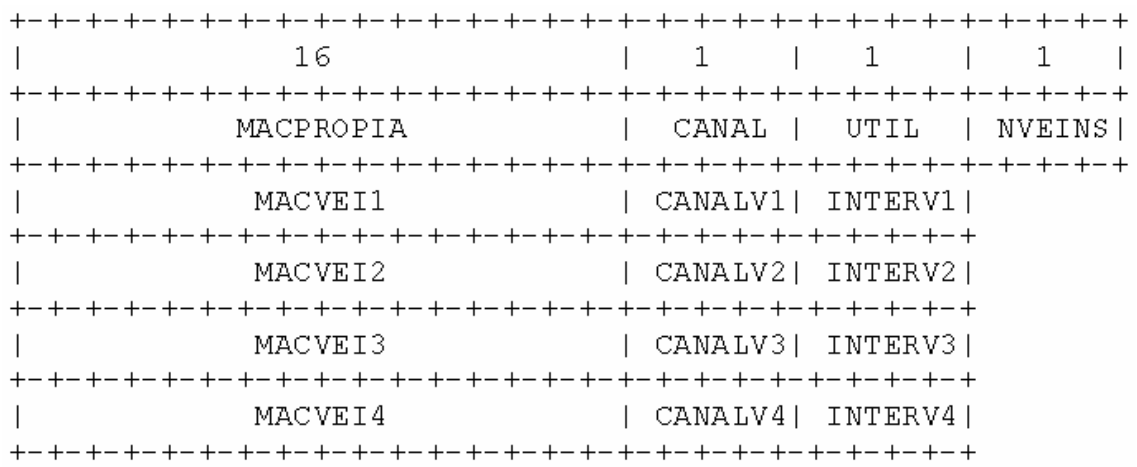


Fig 3.3 Mensaje con información de potencia

Con estos mensajes lo que se quiere es construir unos vectores donde se tenga de manera ordenada toda la información acerca de las interferencias que se crean en los vecinos, y que estos crean en nosotros. Estos mensajes seran llamados a partir de ahora POWER MESSAGE.

El siguiente diagrama, muestra la secuencia que lleva a cabo el Plugin una vez recibe un mensaje. Se debe tener en cuenta que antes de enviar el mensaje el propio Plugin se ha encargado de obtener la información de un fichero ubicado en el sistema y que ha guardado los datos que ahí estaban depositados.

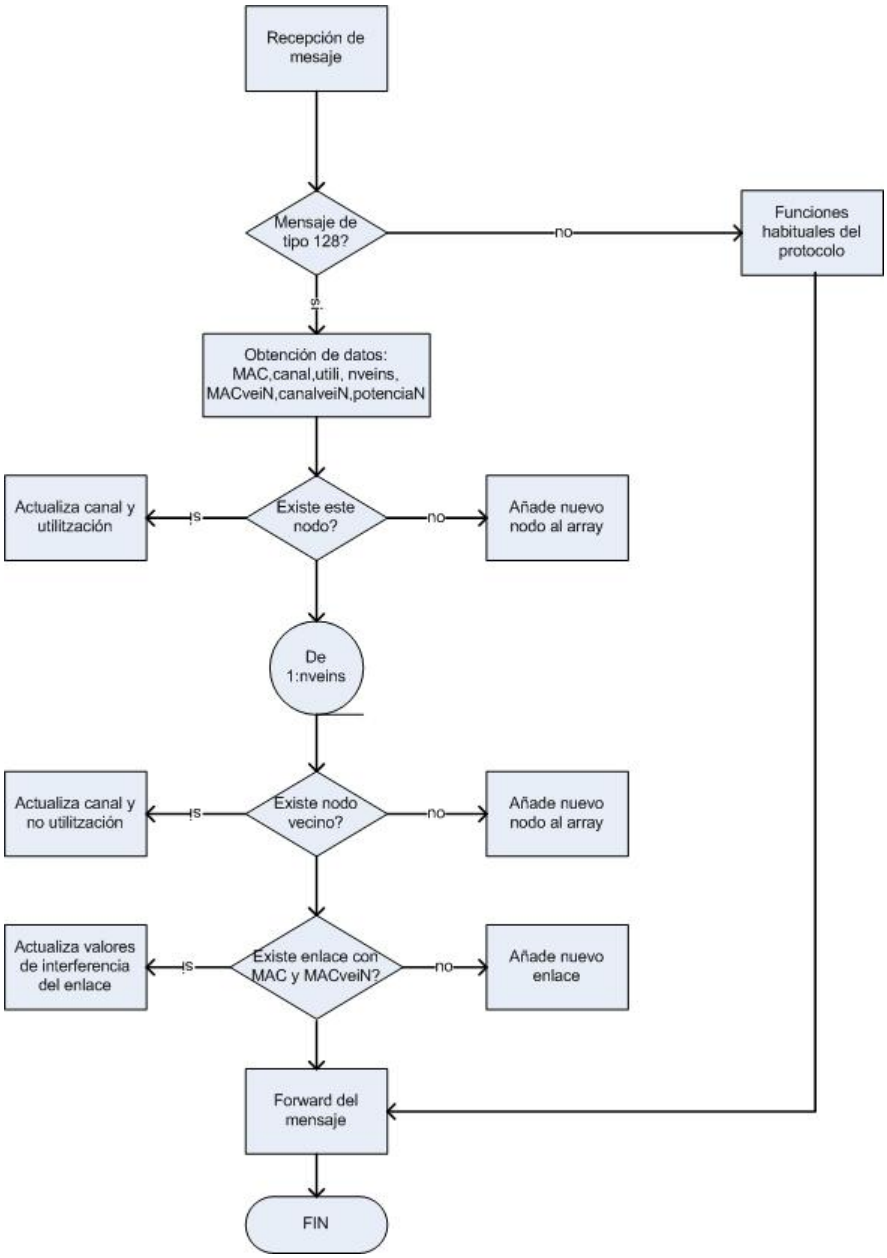


Fig 3.4 Esquema del funcionamiento del script de selección de canal

El sondeo de interferencias y de vecinos se realiza periódicamente cada X segundos, se puede modificar cambiando un valor del script.

Una vez se lleva a cabo el intercambio de mensajes entre todos los nodos que participan en el protocolo, lo que se pretende es conseguir una tabla en la que se tenga la información sobre las interferencias existentes entre cualquier par de nodos y, además, información sobre la utilización y canal de trabajo de cada celda. En la información del enlace no es propiamente dicho información sobre el enlace que une dos nodos, sino que es las interferencias que se crean entre ellos dos,

En el siguiente ejemplo se puede ver el tiempo de convergencia del protocolo, y además la secuencia que siguen los mensajes. El ejemplo propone un escenario en el que hay dos nodos cuya información converge, obteniendo los

valores necesarios en cada uno de los nodos. Una vez la información en ambos nodos han convergido, arranca otro nodo cercano que es escuchado por estos dos nodos, siguiendo el diagrama de secuencia se pueden observar los mensajes que se transmiten entre los dos así como el tiempo que tarda en convergir este nuevo nodo y ser detectado por todos los demás.

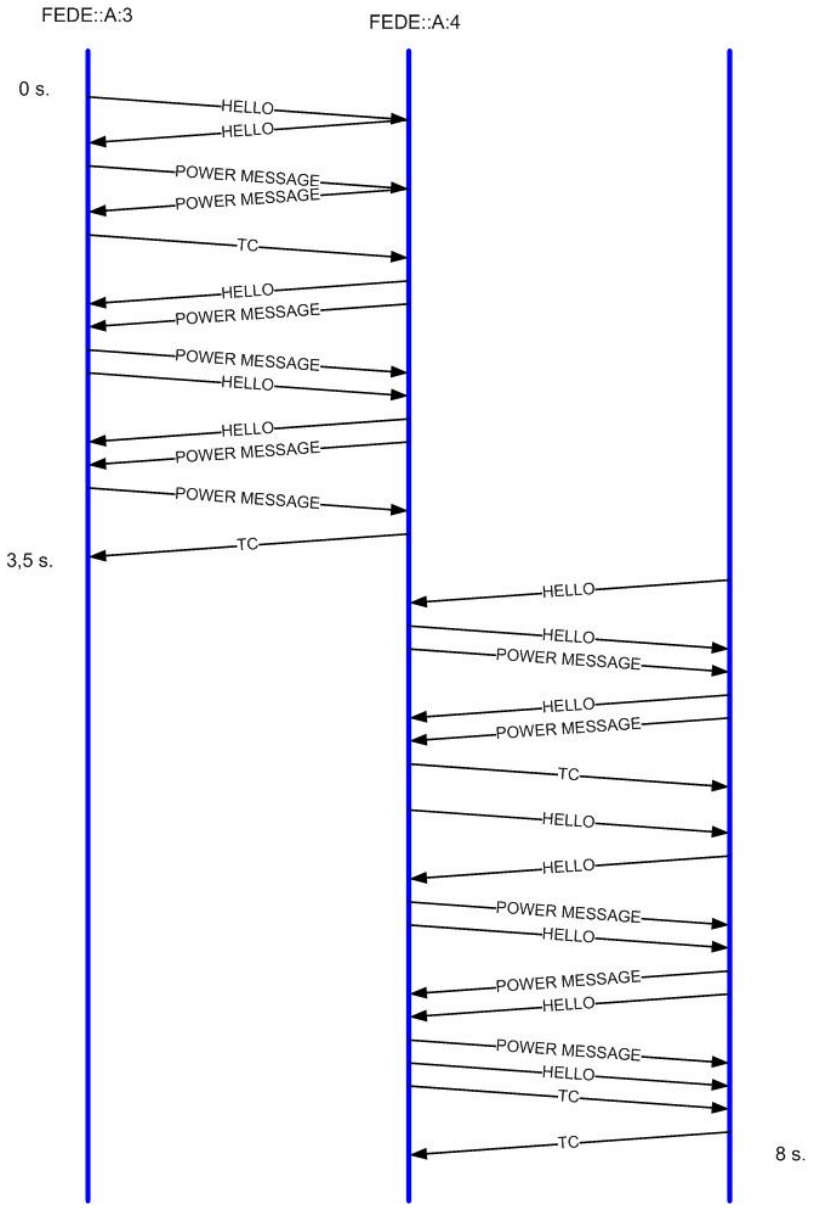


Fig 3.5 Diagrama de secuencia

En el diagrama se debe tener en cuenta que los mensajes del protocolo llegan a todos los nodos a los que tiene alcance, es decir, en el caso de este ejemplo todos los mensajes llegarán a todos los nodos vecinos, aunque en la representación gráfica no se aprecie para poder ver con mayor facilidad la secuencia de mensajes. El dibujo de esta red ejemplo es el siguiente:

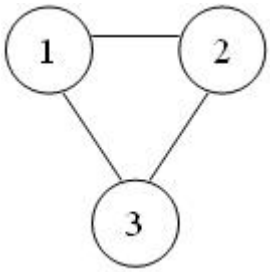


Fig 3.6 Esquema de la red ejemplo

Los mensajes Hello y TC han estado explicados en el apartado que habla sobre el protocolo OLSR. En este mismo apartado se ha visto que información lleva el POWER MESSAGE por lo que se deduce que el mensaje que va del nodo FEDE::A:3 al nodo FEDE::A:4 tiene información de la utilización y el canal de este mismo nodo y la potencia que recibe del nodo FEDE::A:4. Por este motivo a la que se recibe el POWER MESSAGE del nodo vecino se llega a la convergencia porque devuelve la información que faltava para conocer las interferencias entre los dos nodos. Si hubiera más nodos en el alcance para llegar a tener convergencia se debería recibir al menos un mensaje de cada nodo vecino.

Además de la obtención de toda la información necesaria para la determinación de las interferencias, esta información debe ser almacenada para posteriormente ser tratada. Estos datos se guardan en un fichero de texto, donde se guarda el número de nodos con los que trabaja el protocolo de enrutamiento, la utilización de cada nodo y las interferencias que se dan entre los enlaces entre todos estos nodos. Los nodos se deben almacenar en orden y con un índice numérico para poder trabajar con ellos, el nodo 0 siempre es el propio nodo. En las siguientes líneas se puede observar como es el formato de estos documentos:

```
c toda línea que empieza por c indica comentario
c se debe poner una línea que empiece por n, seguida del número de nodos
n 4
c empieza con f, indica los canales en los que trabaja cada nodo por orden del 0 a n-1
f 1 5 8 10
c empieza con u, indica las utilizaciones por orden de los nodos
u 10 23 60 24
c empieza con e, indica los enlaces. Primero se ponen los valores de los nodos que
c crean este enlace (n1 y n2) y posteriormente las interferencias. Primer valor
c interferencia de n1 a n2 y el segundo de n2 a n1
e 0 1 -43 -21
e 0 2 -32 -12
e 0 3 -45 -34
e 1 3 -12 -54
e 2 3 -56 -30
c en la siguiente línea se indica el nodo que ha escrito el fichero, siempre es el 0
--> nodo 0
```

Capítulo 4. Pruebas de la red

Una vez configurados todos los equipos, es decir, tienen todas las interfaces correctamente configuradas, con el protocolo de enrutamiento instalado y trabajando con el plugin que se ha creado, además de todas las otras instalaciones que se han comentado durante este documento, se está preparado para realizar pruebas de funcionamiento.

Se empezará haciendo pruebas con redes de 4 nodos, en las que se cambiará la configuración de la topología para poder demostrar que el plugin funciona correctamente y que se crea una red estable con conectividad entre todos los puntos de esta.

4.1 Prueba 1: topología completa

Esta topología tiene todos los nodos conectados a todos los nodos, es una red mallada en la que todos se ven con todos. Observando las líneas que unen todos los nodos se pueden contar el número de enlaces que se crean con esta topología.

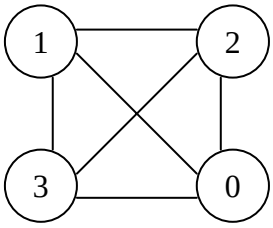


Fig 4.1 Topología mallada

La prueba sigue la siguiente cronología, se tiene trabajando correctamente los nodos 0, 1 y 2, en régimen permanente. Entonces se coloca el nodo 3 en un lugar cercano y una vez ejecutado por primera vez el script de sondeo de canales se procede a arrancar el protocolo de enrutamiento. Desde que arranca hasta que todos los nodos de la red conocen este nuevo nodo, la forma de llegar a él y tienen toda la información que transmite el plugin de potencia transcurren 6.5 segundos. En este momento se puede observar el archivo de texto en el que se guardan los datos y tiene la siguiente salida:

n 4
f 11 12 1 6
u 12 34 20 5
e 0 1 -46 -34
e 0 2 -21 -22
e 0 3 -13 -35
e 3 2 -23 -22
e 3 1 -29 -28
e 1 2 -26 -24

En estas líneas se puede observar como se ha detectado el enlace entre todos los nodos de la red. Se debe tener en cuenta que aunque éste sólo es la salida que nos da un nodo todos los nodos tienen la misma información y un documento igual.

4.2 Prueba 2: topología en línea

En esta prueba se colocan todos los nodos en línea. Es decir, los nodos de la punta sólo conocen el inmediato y los del centro conocen uno a su derecha y otro a su izquierda. En la siguiente figura se aprecia mucho mejor la topología:

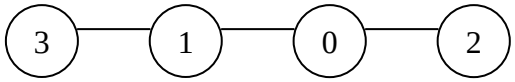


Fig 4.2 Topología en línea

Como se puede observar en esta figura el número de enlaces es menor que en el caso anterior, pero en cambio los mensajes tienen que pasar por más nodos para llegar a su destino. La prueba sigue el mismo método que la anterior, se parte con los nodos 0, 1 y 3 iniciados y se lanza el nodo 2, en ese momento se intercambian los mensajes. Aunque el nodo 0 rápidamente se percata del nuevo nodo no es hasta 8.5 segundos después que se puede decir que la red ha convergido y que todos los nodos tienen toda la información de potencia necesaria.

Comparado a la prueba anterior el tiempo es mayor, esto es debido a que los mensajes en lugar de llegar en el mismo instante a todos los nodos, primero llegan a uno y este nodo reenvía su tabla de potencias que debe pasar antes por otro nodo intermedio.

El resultado del fichero de datos es el siguiente:

n 4
f 11 12 13 7
u 12 34 20 5
e 0 1 -21 -22
e 0 2 -23 -19
e 1 3 -14 -10

Con estos datos salta a la vista lo que se comentó con anterioridad, esta red tiene muchos menos enlaces. Podemos comprobar que el funcionamiento del plugin parece correcto.

4.3 Prueba 3: nodo aislado

En esta topología se tiene tres nodos conectados entre sí, todos los nodos se ven entre ellos, además se tiene un nodo aislado que sólo es visible por uno de los nodos anteriormente comentados.

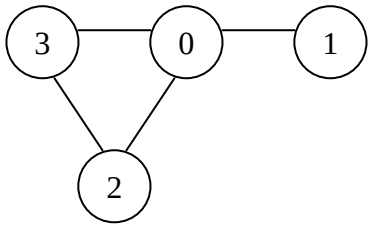


Fig 4.3 Topología con nodo aislado

Después de ver la topología, se puede intuir que se pondrán en funcionamiento los nodos 0, 2 y 3 y cuando estos estén en convergencia se pondrá en marcha el nodo 1. En esta topología el número de enlaces llega a 4. Es una situación intermedia respecto a las situaciones anteriores ya que una vez el nodo 0 conoce el nuevo nodo, la información llega a los otros dos nodos al mismo tiempo. Por este motivo el tiempo de convergencia es también de un valor intermedio, son 7 segundos. El fichero de salida que se obtiene es el siguiente:

```
n 4
f 12 6 9 1
u 12 34 20 5
e 0 1 -46 -34
e 0 2 -11 -22
e 0 3 -29 -35
e 3 2 -23 -25
```

4.4 Prueba 4: casi mallada

Esta topología es casi mallada, debido a que tiene enlaces entre todos los nodos que forman la red excepto entre dos nodos, es decir, sólo falta un enlace. La metodología de la prueba es muy parecida que en la topología mallada, se ponen en funcionamiento los nodos 0,2 y 3. Cuando están trabajando correctamente se lanza el nodo 1, esto se hace porque se quiere que el último nodo en ponerse en funcionamiento sea uno que no tiene un enlace hacia todos los nodos de la red.

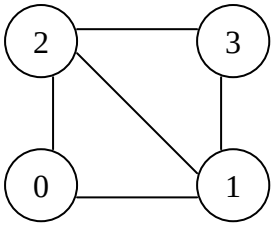


Fig 4.4 Topología casi mallada

Una vez se ponen en marcha el nodo 1, tarda 8.3 segundos en llegar a converger toda la red. El fichero de salida que se obtiene es el siguiente:

n 4
f 12 9 1 5
u 12 34 20 5
e 0 1 -21 -13
e 0 2 -23 -15
e 2 1 -16 -19
e 2 3 -11 -22
e 1 3 -19 -10

Capítulo 5. Conclusiones

Este proyecto tiene como mayor finalidad la de dotar a ordenadores personales, del hardware y software necesario para convertirlos en APs inalámbricas inteligentes, con capacidad para trabajar en las mejores condiciones y con acceso a la red que forman entre todos los nodos.

Este TFC trabajará con tecnología inalámbricas que cumple todas las normativas exigidas en materia de medio ambiente. Se utiliza la tecnología 802.11b que trabaja en un rango de frecuencias libre (2.4Ghz), por lo que se puede trabajar en él según las leyes vigentes (Ley General de las Telecomunicaciones). Este tipo de redes además de cumplir las normativas tampoco dependen de cableado ninguno por lo que no se deben crear grandes infraestructuras en materia comunicación, aunque si de electricidad.

El objetivo de este proyecto es el de crear una red inalámbricas mallada, en la que todos los nodos que aparezcan puedan trabajar en ella con la mayor calidad posible y sin entorpecer el trabajo de los otros nodos. Para llegar a este objetivo final se ha desarrollado el proyecto mediante pequeños hitos para poder llegar a la solución final.

El primer objetivo fue el de obtener unos ordenadores con las capacidades físicas y de software que se necesitaban. Para ello se debían dotar a los ordenadores de tecnología inalámbricas, en este caso se les ha dado tarjetas PCMCIA. Además algunas de estas tarjetas deben trabajar en modo AP por lo que se les ha tenido que instalar un software adicional: Hostap. Esta red se deseaba que trabajara con IPv6, por lo que se tuvo que dotar a todos los nodos de esta capacidad.

A partir de este momento, el siguiente objetivo fue el de hacer que los nodos fueran inteligentes. Para ello se empezó con darles la capacidad de seleccionar el canal en el que debían trabajar. Mediante sondeos del medio y algoritmos de asignación cada nodo puede elegir el canal en el que trabajará y tendrá menos interferencias de nodos vecinos.

Una vez cumplido esto el próximo objetivo para la creación de una red, es la de comunicar todos los nodos. En este sentido el primer paso era el de conocer las tecnologías actuales y tener una base teórica acerca de los protocolos de enrutamiento inalámbricas que hay en la actualidad. Para ello se buscó información acerca de las características de estos protocolos, y de los principales, así como sus ventajas e inconvenientes. Una vez se tuvo el conocimiento imprescindible se pudo tomar la decisión y se eligió uno de ellos. Se decidió usar el protocolo OLSR, ya que por sus características. Tanto el tamaño de las redes que puede abarcar, como el número de retransmisiones que ofrecía era el idóneo. Además de contar con una implementación perfectamente testeada que nos permitía centrarnos en nuestro desarrollo.

Con este protocolo se dotaba a la red de comunicación entre puntos. Pero además de la selección individual del canal de trabajo por cada nodo de la red,

también se pretende que cada nodo tenga información de los nodos vecinos así como de la interferencia que este mismo nodo causa en los demás. Esta información será utilizada para que cada nodo pueda seleccionar el canal en el que trabaja con la mayor información posible. De este modo las interferencias con los vecinos disminuirán lo que repercutirá muy positivamente en el rendimiento de la red. Esta información se deseaba que fuera transmitida de la manera más eficiente posible por lo que para poder enviarla se usó el protocolo de enrutamiento. Mediante un plugin se consiguió que el protocolo obtuviera la información necesaria acerca de los vecinos y las interferencias que estos creaban. Esta información se retransmitía de forma óptima hacia los vecinos que tuvieran visibilidad con ellos. Esto se hacia mediante la creación de otro tipo de mensaje para el protocolo de enrutamiento.

Todos estos objetivos y los pasos seguidos para su consecución está documentado en este proyecto para de esto modo facilitar las posibles líneas futuras que en el siguiente párrafo se detallan.

Este proyecto es la continuación del proyecto realizado por Albert Bertran Roman (Puntos de acceso inteligentes basados en Linux (I)). Por lo que antes de empezar con este proyecto se tenía ya información y esta implementado el algoritmo de selección de canal. Además de todo ello también se había desarrollado un script en el que cada nodo conocía el troughput que tenía y el rendimiento al que estaba trabajando. Estos scripts a lo largo de este proyecto se han adecuado y modificado a las necesidades de este. Una línea futura sería la de utilizar los dos proyectos de una manera totalmente conjunta para poder hacer pruebas de rendimiento de la red, pruebas de carga y en la medida de lo posible probarlo en explotaciçon. Otra línea de acción sería la de dotar de seguridad a la red mediante mecanismo WEP, WPA o listas de acceso a los APs, de este modo se podría tener un control mayor de los usuarios que se tienen si es un lugar público o si la información que maneja al red es confidencial. Además otra línea futura seria el uso del sistema de señalización OLSR con otros mecanismos de gestión distribuida, como es el control de carga implementado en el proyecto anteriormente mencionado.

Además de esto también seria muy interesante un análisis más profundo del rendimiento del protocolo mediante simulaciones. De este modo se podría comprobar el tiempo de convergencia, cantidad de información intercambiada, a medida que el número de nodos aumenta o el tráfico que circula por la red es distinto.

Capítulo 6. Referencias y bibliografía

Universidad Carnegie-Mellon, http://www.cmu.edu/computing/wireless/. Junio, 2005.

Host AP driver for Intersil Prism2/2.5/3, hostapd, and WPA Supplicant, http://hostap.epitest.fi/. Setiembre, 2005.

Kernel Howto, www.**linux**docs.org/HOWTOs/Kernel-HOWTO.html. Marzo, 2005.

Jean Tourrilhes, Wireless Tools for Linux, http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html. Marzo, 2005.

Andreas Tonnesen, OLSR, www.olsr.org. Mayo, 2005.

Andreas Tonnesen, UniK olsrd plugin implementation HOWTO, http://www.olsr.org/docs/olsrd-plugin-howto.html. Junio, 2005.

Ian F. Akyldiz, Xudong Wang, Weilin Wang. Wireless Mesh networks: a survey. Junio, 2004.

R. Oiger. MANET Extensión of OSPF usign CDS Flooding. Febrero, 2005.

T. Henderson, OSPFv2 Wireless Interface Type. Mayo, 2004.

Andreas Tonnesen, Andreas Hafslund, Oivind Kure, The Unik – OLSR plugin library. Enero, 2005.

Andreas Tonnesen. Unik olsrd plugin implementation howto. Marzo, 2004.

T. Clausen, P. Jacquet. Optimizad Link State Routing Protocol (OLSR). RFC 3626. Octubre, 2003.

C. Perkins. Ad hoc On-Demmand Distance Vector (AODV) Routing. RFC 3561. Julio, 2003.

Jean-Yves Le Boudec. IPv6. Enero, 2003.

Albert Beltran Roman. Puntos de acceso inteligentes basados en linux (I). Febrero, 2005.

Peter Bieringer, Linux IPv6 Howto. Enero, 2005.

Marcos García Martí. Programación de un protocolo multicast geográfico sobre IPv6. Setiembre, 2004.

Daniel Lang. A Comprehensive overview about selected Ad hoc Networking Routing Protocols. Marzo, 2003.

David Pérez Álvarez. Estudi i avaluació del traspàs en xarxes IP: IEEE 802.11 i Mobile IP. Julio, 2003.

Alexis Porro Pérez. Despliegue de WLANs en exteriores: desarrollo de una herramienta para la toma de medidas georeferenciadas. Julio, 2003.

Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS

TÍTOL DEL TFC/PFC

TITULACIÓ:

AUTOR: Manuel Pérez Pérez

DIRECTOR: José González González

DATA: 25 de febrer de 2003

Anexo I. Código plugin para OLSR

```
/*
 * Copyright (c) 2004, Andreas TÃ¸nnesen(andreto-at-olsr.org)
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * * Redistributions of source code must retain the above copyright
notice,
 *   this list of conditions and the following disclaimer.
 * * Redistributions in binary form must reproduce the above copyright
notice,
 *   this list of conditions and the following disclaimer in the
documentation
 *   and/or other materials provided with the distribution.
 * * Neither the name of the UniK olsr daemon nor the names of its
contributors
 *   may be used to endorse or promote products derived from this
software
 *   without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.
 * IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY
 * OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 */

/* $Id: olsrd_power.c,v 1.6 2005/03/04 22:03:54 kattemat Exp $ */

/*
 * Dynamic linked library example for UniK OLSRd
 */

#include "olsrd_power.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
```

```
#ifdef OS
#undef OS
#endif

#ifdef WIN32
#define close(x) closesocket(x)
#define OS "Windows"
#endif
#ifdef linux
#define OS "GNU/Linux"
#endif
#ifdef __FreeBSD__
#define OS "FreeBSD"
#endif

#ifndef OS
#define OS "Undefined"
#endif

int ipc_socket;
int ipc_open;
int ipc_connection;
int ipc_connected;

int
ipc_send(char *, int);

#ifdef WIN32

static char *inet_ntop4(const unsigned char *src, char *dst, int size)
{
  static const char fmt[] = "%u.%u.%u.%u";
  char tmp[sizeof "255.255.255.255"];

  if (sprintf(tmp, fmt, src[0], src[1], src[2], src[3]) > size)
    return (NULL);

  return strcpy(dst, tmp);
}

char *inet_ntop(int af, void *src, char *dst, int size)
{
  switch (af)
  {
  case AF_INET:
    return (inet_ntop4(src, dst, size));

  default:
    return (NULL);
  }
}

#endif

/**
 *Do initialization here
 *
 *This function is called by the my_init
 *function in uolsrd_plugin.c
 */
int
```

```
olsr_plugin_init()
{
  int i;
  struct olsr_apm_info apm_info;
  numeronodes=0;
  numerolinks=0;
  if(ipversion != AF_INET)
    {
      fprintf(stderr, "This plugin only supports IPv6!\n");
    }
  /* Initial IPC value */
  ipc_open = 0;

  /* Init list */
  for(i = 0; i < HASHSIZE; i++)
    {
      list[i].next = &list[i];
      list[i].prev = &list[i];
    }

  if(apm_read(&apm_info) < 0)
    {
      has_apm = 0;
      olsr_printf(1, "No APM info avalible! This node will not
generate powermessages!\n\n");
    }
  else
    {
      olsr_printf(1, "Node has APM info!\n");
      has_apm = 1;
    }

  /* Register functions with olsrd */
  olsr_parser_add_function(&olsr_parser, PARSER_TYPE, 1);

  olsr_register_timeout_function(&olsr_timeout);

  olsr_register_scheduler_event(&olsr_event, NULL, EMISSION_INTERVAL,
0, NULL);

  return 1;
}

int
plugin_ipc_init()
{
  struct sockaddr_in6 sin;
  olsr_u32_t yes = 1;

  /* Init ipc socket */
  if ((ipc_socket = socket(AF_INET6, SOCK_STREAM, 0)) == -1)
    {
      perror("IPC socket");
      return 0;
    }
  else
    {
      if (setsockopt(ipc_socket, SOL_SOCKET, SO_REUSEADDR, (char
*)&yes, sizeof(yes)) < 0)
      {
      perror("SO_REUSEADDR failed");
```

```
      return 0;
      }

#ifdef __FreeBSD__
      if (setsockopt(ipc_socket, SOL_SOCKET, SO_NOSIGPIPE, (char
*)&yes, sizeof(yes)) < 0)
      {
      perror("SO_REUSEADDR failed");
      return 0;
      }
#endif

      /* Bind the socket */

      /* complete the socket structure */
      memset(&sin, 0, sizeof(sin));
      sin.sin6_family = AF_INET;
      sin.sin6_addr = in6addr_any;
      sin.sin6_port = htons(IPC_PORT);

      /* bind the socket to the port number */
      if (bind(ipc_socket, (struct sockaddr *) &sin, sizeof(sin)) == -
1)
      {
        perror("IPC bind");
        return 0;
      }

      /* show that we are willing to listen */
      if (listen(ipc_socket, 1) == -1)
      {
        perror("IPC listen");
        return 0;
      }

      /* Register with olsrd */
      add_olsr_socket(ipc_socket, &ipc_action);

    }

  ipc_open = 1;
  return 1;
}

void
ipc_action(int fd)
{
  struct sockaddr_in6 pin;
  socklen_t addrlen;
  char *addr;

  addrlen = sizeof(struct sockaddr_in6);

  if ((ipc_connection = accept(ipc_socket, (struct sockaddr *)  &pin,
&addrlen)) == -1)
    {
      perror("IPC accept");
      exit(1);
    }
  else
    {
```

```
      inet_ntop(AF_INET6,&pin.sin6_addr,addr,128);
        ipc_connected = 1;
        olsr_printf(1, "POWER: Connection from %s\n",addr);
    }

}

/*
 * destructor - called at unload
 */
void
olsr_plugin_exit()
{
  if(ipc_open)
    close(ipc_socket);
}



/* Mulitpurpose funtion */
int
plugin_io(int cmd, void *data, size_t size)
{

  switch(cmd)
    {
    default:
      return 0;
    }

  return 1;
}



/**
 *A timeoutfunction called every time
 *the scheduler is polled
 */
void
olsr_timeout()
{
  struct pwrentry *tmp_list;
  struct pwrentry *entry_to_delete;
  int index;


  for(index=0;index<HASHSIZE;index++)
    {
      tmp_list = list[index].next;
      /*Traverse MID list*/
      while(tmp_list != &list[index])
      {
        /*Check if the entry is timed out*/
        if(olsr_timed_out(&tmp_list->timer))
          {
            entry_to_delete = tmp_list;
            tmp_list = tmp_list->next;
            olsr_printf(1, "POWER info for %s timed out.. deleting
it\n",
```

```
                  olsr_ip_to_string(&entry_to_delete->originator));
            /* Dequeue */
            entry_to_delete->prev->next = entry_to_delete->next;
            entry_to_delete->next->prev = entry_to_delete->prev;

            /* Delete */
            free(entry_to_delete);
          }
        else
            tmp_list = tmp_list->next;
      }
    }

  return;
}


/**
 *Scheduled event
 */
void
olsr_event(void *foo)
{
  union olsr_message *message = (union olsr_message*)buffer;
  struct interface *ifn;

  /* If we can't produce power info we do nothing */
  if(!has_apm)
    return;

  olsr_printf(3, "PLUG-IN: Generating package - ");

  /* looping trough interfaces */
  for (ifn = ifs; ifn ; ifn = ifn->int_next)
    {
      olsr_printf(3, "[%s]  ", ifn->int_name);
      /* Fill message */
      if(ipversion == AF_INET)
      {
        /* IPv4 */
        message->v4.olsr_msgtype = MESSAGE_TYPE;
        message->v4.olsr_vtime = double_to_me(7.5);
        message->v4.olsr_msgsize = htons(sizeof(struct olsrmsg));
        memcpy(&message->v4.originator, main_addr, ipsize);
        message->v4.ttl = MAX_TTL;
        message->v4.hopcnt = 0;
        message->v4.seqno = htons(get_msg_seqno());

        get_powerstatus(&message->v4.msg);

        if(net_outbuffer_push(ifn, (olsr_u8_t *)message, sizeof(struct
olsrmsg)) != sizeof(struct olsrmsg))
          {

            /* Send data and try again */
            net_output(ifn);
            if(net_outbuffer_push(ifn, (olsr_u8_t *)message,
sizeof(struct olsrmsg)) != sizeof(struct olsrmsg))
           olsr_printf(1, "Powerplugin: could not write to buffer for
interface: %s\n", ifn->int_name);
          }
```

```
      }
      else
      {
        /* IPv6 */
        message->v6.olsr_msgtype = MESSAGE_TYPE;
        message->v6.olsr_vtime = double_to_me(7.5);
        message->v6.olsr_msgsize = htons(sizeof(struct olsrmsg));
        memcpy(&message->v6.originator, main_addr, ipsize);
        message->v6.ttl = MAX_TTL;
        message->v6.hopcnt = 0;
        message->v6.seqno = htons(get_msg_seqno());
        get_powerstatus(&message->v6.msg);
        if(net_outbuffer_push(ifn, (olsr_u8_t *)message, sizeof(struct
olsrmsg6)) != sizeof(struct olsrmsg6))
          {
            /* Send data and try again */
            net_output(ifn);
            if(net_outbuffer_push(ifn, (olsr_u8_t *)message,
sizeof(struct olsrmsg6)) != sizeof(struct olsrmsg6))
            olsr_printf(1, "Powerplugin: could not write to buffer for
interface: %s\n", ifn->int_name);
          }

      }

    }
  olsr_printf(2, "\n");

  /* Try to set up IPC socket if not already up */
  if(!ipc_open)
    plugin_ipc_init();

  print_power_table();

  return;
}



void
olsr_parser(union olsr_message *m, struct interface *in_if, union
olsr_ip_addr *in_addr)
{
  struct  powermsg *message;
  union olsr_ip_addr originator;
  double vtime;

  /* Fetch the originator of the messsage */
  memcpy(&originator, &m->v6.originator, ipsize);

  /* Fetch the message based on IP version */
  if(ipversion == AF_INET)
    {
      message = &m->v4.msg;
      vtime = me_to_double(m->v4.olsr_vtime);
    }
  else
    {
      message = &m->v6.msg;
      vtime = me_to_double(m->v6.olsr_vtime);
```

```
    }

  /* Check if message originated from this node */
  if(memcmp(&originator, main_addr, ipsize) == 0){
    /* If so - back off */
    return;}

  /* Check that the neighbor this message was received
     from is symmetric */
  if(check_neighbor_link(in_addr) != SYM_LINK)
    {
      /* If not symmetric - back off */
      olsr_printf(3, "Received POWER from NON SYM neighbor %s\n",
olsr_ip_to_string(in_addr));
      return;
    }

  /* Check if this message has been processed before
   * Remeber that this also registeres the message as
   * processed if nessecary
   */
  if(!check_dup_proc(&originator,
                     ntohs(m->v6.seqno))) /* REMEMBER NTOHS!! */
    {
      /* If so - do not process */
      goto forward;
    }

  /* Process */

  olsr_printf(3, "POWER PLUG-IN: Processing PWR from %s seqno: %d\n",
            olsr_ip_to_string(&originator),
            ntohs(m->v6.seqno));
  /* Call a function that updates the database entry */
  update_power_entry(&originator, message, vtime);


 forward:
  /* Forward the message if nessecary
   * default_fwd does all the work for us!
   */
  default_fwd(m,
              &originator,
              ntohs(m->v6.seqno), /* IMPORTANT!!! */
              in_if,
              in_addr);
}



/**
 *Update or register a new power entry
 */
int
update_power_entry(union olsr_ip_addr *originator, struct powermsg
*message, double vtime)
{
  int hash;
  struct pwrentry *entry;
  int contador=0;
  int numero;
```

```
  hash = olsr_hashing(originator);
  /* Check for the entry */
  for(entry = list[hash].next;
      entry != &list[hash];
      entry = entry->next)
    {
        memcpy(entry->mac,message->mac,strlen(message->mac));
        entry->mac[17]='\0';
      entry->canal = message->canal;
      entry->util = message->util;
      entry->nveins = message->nveins;
        numeronodes=addnode(message->mac,message->canal,message-
>util,numeronodes);
      numero=numeronodes;

      if(contador<entry->nveins){
        memcpy(entry->macv2,message->macv2,strlen(message->macv2));
        entry->macv2[17]='\0';
      entry->canalv2 = message->canalv2;
      entry->potenciarebudav2 = message->potenciarebudav2;
        contador++;
        numeronodes=addnodevei(message->mac,message->macv2,message-
>canalv2,message->potenciarebudav2,numeronodes);
      }
      else{
      entry->macv2[17]='\0';
    }

      if(contador<entry->nveins){
        memcpy(entry->macv3,message->macv3,strlen(message->macv3));
        entry->macv3[17]='\0';
      entry->canalv3 = message->canalv3;
      entry->potenciarebudav3 = message->potenciarebudav3;
        contador++;
        numeronodes=addnodevei(message->mac,message->macv3,message-
>canalv3,message->potenciarebudav3,numeronodes);
      }
      else{
      entry->macv3[17]='\0';
    }

    if(contador<entry->nveins){
      memcpy(entry->macv4,message->macv4,strlen(message->macv4));
      entry->macv4[17]='\0';
      entry->canalv4 = message->canalv4;
      entry->potenciarebudav4 = message->potenciarebudav4;
      contador++;
      numeronodes=addnodevei(message->mac,message->macv4,message-
>canalv4,message->potenciarebudav4,numeronodes);
    }
      else{
      entry->macv4[17]='\0';
    }

      olsr_get_timestamp(vtime * 1000, &entry->timer);
      return 0;
  }

    int contem=0;
      if(contem<nveinsprop){
```

```
      numeronodes=addnodevei(macpropia,macpropv2,canalpropv2,potenciar
ebudapropv2,numeronodes);
            contem++;
      }
      if(contem<nveinsprop){

      numeronodes=addnodevei(macpropia,macpropv3,canalpropv3,potenciar
ebudapropv3,numeronodes);
            contem++;
      }
        if(contem<nveinsprop){

numeronodes=addnodevei(macpropia,macpropv4,canalpropv4,potenciarebudap
ropv4,numeronodes);
               contem++;
        }

    olsr_printf(1, "New power entry %s: ",
olsr_ip_to_string(originator));

    olsr_printf(1, "Canal: %d utilitzacio %d \n",message-
>canal,message->util);

    entry = olsr_malloc(sizeof(struct pwrentry), "POWERPLUGIN: new
power entry");

  /* Fill struct */

  memcpy(&entry->originator, originator, ipsize);
  entry->canal = message->canal;
  olsr_get_timestamp(vtime * 1000, &entry->timer);

  /* Queue */
  entry->next = list[hash].next->prev;
  entry->prev = &list[hash];
  list[hash].next->prev = entry;
  list[hash].next = entry;

  return 1;
}


/**
 *Print all registered power entries
 */

void
print_power_table()
{
  int hash;
  int r=0;
  int intro=0;
  struct pwrentry *entry;
  char buf[200];
  char mac[18];
  int canal;
  int utilitzacio;
  if(!ipc_connection)
    return;
```

```
  ipc_send("--POWERTABLE--\n", 15);

  for(hash = 0; hash < HASHSIZE; hash++)
    /* Check for the entry */
    for(entry = list[hash].next;
      entry != &list[hash];
      entry = entry->next)
      {
      sprintf(buf, "[%s]: ", olsr_ip_to_string(&entry->originator));
      ipc_send(buf, strlen(buf));
      sprintf(buf,"INFORMACIO DE NODES \n");
    ipc_send(buf, strlen(buf));
    int stop=0;
      while(stop<numeronodes){
            sprintf(buf,"Node %d: mac: %s | canal: %d | utilitzacio: %d
\n",stop+1,nodes[stop].mac,nodes[stop].canal,nodes[stop].utilitzacio);
                  ipc_send(buf, strlen(buf));
            stop++;
      }

      sprintf(buf,"INFORMACIO DE LINKS \n");
    ipc_send(buf, strlen(buf));
      stop=0;
    while(stop<numerolinks){
        sprintf(buf,"Link %d: mac1: %s | mac2: %s | interferencia 1 a
2: %d | interferencia 2 a 1: %d
\n",stop+1,enlaces[stop].node1mac,enlaces[stop].node2mac,enlaces[stop]
.int12,enlaces[stop].int21);
        ipc_send(buf, strlen(buf));
        stop++;
    }
  }

      ipc_send("--------------\n", 15);

}

int
ipc_send(char *data, int size)
{
  if(!ipc_connected)
    return 0;

#ifdef __FreeBSD__
  if (send(ipc_connection, data, size, 0) < 0)
#else
  if (send(ipc_connection, data, size, MSG_NOSIGNAL) < 0)
#endif
    {
      //perror("send - IPC");
      olsr_printf(1, "(OUTPUT)IPC connection lost!\n");
      close(ipc_connection);
      //use_ipc = 0;
      ipc_connected = 0;
      return -1;
    }

  return 1;
}

/**
```

```
 *Fill a powermsg struct with power data
 */
int
get_powerstatus(struct powermsg *msg)
{
  int word=0;
  int canal,utilitzacio;
  int nveins=0;
  nveinsprop=0;
  char address[18];
  char address1[18];
  char address2[18];
  char address3[18];
  char address4[18];
  FILE *fd_iwconfigres;
        fd_iwconfigres=fopen("/root/Albert/scripts/aps","r");
        if (fd_iwconfigres==NULL)
        {
                fprintf(stderr, "No se puede abrir el fichero de
datos.\n");
                return 1;
        }
        int potencia;
        word=fscanf(fd_iwconfigres,"%d",&utilitzacio);
        msg->util=utilitzacio;
            word=fscanf(fd_iwconfigres,"%s",address);
        memcpy(msg->mac,address,strlen(address));
            msg->mac[17]='\0';
            memcpy(macpropia,address,strlen(address));
        macpropia[17]='\0';
            msg->canal=12;
            numeronodes=addnode(address,msg->canal,msg-
>util,numeronodes);
        word=fscanf(fd_iwconfigres,"%s",address2);

        if((strlen(address2)==17)&&(word!=EOF)){
            memcpy(msg->macv2,address2,strlen(address2));
                  msg->macv2[17]='\0';
                  memcpy(macpropv2,address2,strlen(address2));
            macpropv2[17]='\0';
                  word=fscanf(fd_iwconfigres,"%s",address2);
            word=fscanf(fd_iwconfigres,"%d",&canal);
            msg->canalv2=canal;
                  canalpropv2=canal;
            word=fscanf(fd_iwconfigres," %d",&potencia);
            msg->potenciarebudav2=potencia;
                  potenciarebudapropv2=potencia;
                  numeronodes=addnodevei(msg-
>mac,macpropv2,canalpropv2,potenciarebudapropv2,numeronodes);
            nveinsprop++;
                  nveins++;
          }
              else{
                  memcpy(msg->macv2,address,strlen(address));
            msg->macv2[17]='\0';
                  msg->canalv2=0;
                  msg->potenciarebudav2=0;
              }
            word=fscanf(fd_iwconfigres,"%s",address3);

        if((strlen(address3)==17)&&(word!=EOF))
```

```
        {
            memcpy(msg->macv3,address3,strlen(address3));
            msg->macv3[17]='\0';
          memcpy(macpropv3,address3,strlen(address3));
          macpropv3[17]='\0';
              word=fscanf(fd_iwconfigres,"%s",address2);
          word=fscanf(fd_iwconfigres,"%d",&canal);
          msg->canalv3=canal;
            canalpropv3=canal;
          word=fscanf(fd_iwconfigres," %d",&potencia);
          msg->potenciarebudav3=potencia;
            potenciarebudapropv3=potencia;
            numeronodes=addnodevei(msg-
>mac,macpropv3,canalpropv3,potenciarebudapropv3,numeronodes);
            nveinsprop++;
          nveins++;
        }
            else{
                  memcpy(msg->macv3,address,strlen(address));
                  msg->macv3[17]='\0';
                  msg->canalv3=0;
                  msg->potenciarebudav3=0;
        }

            word=fscanf(fd_iwconfigres,"%s",address4);

        if((strlen(address4)==17)&&(word!=EOF))
        {
              memcpy(msg->macv4,address4,strlen(address4));
              msg->macv4[17]='\0';
              memcpy(macpropv4,address4,strlen(address4));
              macpropv4[17]='\0';
                    word=fscanf(fd_iwconfigres,"%s",address2);
              word=fscanf(fd_iwconfigres,"%d",&canal);
              msg->canalv4=canal;
              canalpropv4=canal;
              word=fscanf(fd_iwconfigres," %d",&potencia);
              msg->potenciarebudav4=potencia;
              potenciarebudapropv4=potencia;
              numeronodes=addnodevei(msg-
>mac,macpropv4,canalpropv4,potenciarebudapropv4,numeronodes);
              nveinsprop++;
              nveins++;
        }
            else{
              memcpy(msg->macv4,address,strlen(address));
              msg->macv4[17]='\0';
              msg->canalv4=0;
              msg->potenciarebudav4=0;
        }

            fclose(fd_iwconfigres);
        msg->nveins=nveins;
            return 1;
}


/*************************************************************
 *                 TOOLS DERIVED FROM OLSRD                  *
 *************************************************************/
```

```
/**
 *Hashing function. Creates a key based on
 *an 32-bit address.
 *@param address the address to hash
 *@return the hash(a value in the 0-31 range)
 */
olsr_u32_t
olsr_hashing(union olsr_ip_addr *address)
{
  olsr_u32_t hash;
  char *tmp;

  if(ipversion == AF_INET)
    /* IPv4 */
    hash = (ntohl(address->v4));
  else
    {
      /* IPv6 */
      tmp = (char *) &address->v6;
      hash = (ntohl(*tmp));
    }

  //hash &= 0x7fffffff;
  hash &= HASHMASK;

  return hash;
}



/**
 *Checks if a timer has times out. That means
 *if it is smaller than present time.
 *@param timer the timeval struct to evaluate
 *@return positive if the timer has not timed out,
 *0 if it matches with present time and negative
 *if it is timed out.
 */
int
olsr_timed_out(struct timeval *timer)
{
  return(timercmp(timer, now, <));
}



/**
 *Initiates a "timer", wich is a timeval structure,
 *with the value given in time_value.
 *@param time_value the value to initialize the timer with
 *@param hold_timer the timer itself
 *@return nada
 */
void
olsr_init_timer(olsr_u32_t time_value, struct timeval *hold_timer)
{
  olsr_u16_t  time_value_sec;
  olsr_u16_t  time_value_msec;

  time_value_sec = time_value/1000;
```

```
  time_value_msec = time_value-(time_value_sec*1000);

  hold_timer->tv_sec = time_value_sec;
  hold_timer->tv_usec = time_value_msec*1000;
}




/**
 *Generaties a timestamp a certain number of milliseconds
 *into the future.
 *
 *@param time_value how many milliseconds from now
 *@param hold_timer the timer itself
 *@return nada
 */
void
olsr_get_timestamp(olsr_u32_t delay, struct timeval *hold_timer)
{
  olsr_u16_t  time_value_sec;
  olsr_u16_t  time_value_msec;

  time_value_sec = delay/1000;
  time_value_msec= delay - (delay*1000);

  hold_timer->tv_sec = now->tv_sec + time_value_sec;
  hold_timer->tv_usec = now->tv_usec + (time_value_msec*1000);
}


/**
 *Converts a olsr_ip_addr to a string
 *Goes for both IPv4 and IPv6
 *
 *NON REENTRANT! If you need to use this
 *function twice in e.g. the same printf
 *it will not work.
 *You must use it in different calls e.g.
 *two different printfs
 *
 *@param the IP to convert
 *@return a pointer to a static string buffer
 *representing the address in "dots and numbers"
 *
 */
char *
olsr_ip_to_string(union olsr_ip_addr *addr)
{

  char *ret;
  struct in_addr in;

  if(ipversion == AF_INET)
    {
      in.s_addr=addr->v4;
      ret = inet_ntoa(in);
    }
  else
    {
```

```
      /* IPv6 */
      ret = (char *)inet_ntop(AF_INET6, &addr->v6, ipv6_buf,
sizeof(ipv6_buf));
    }

  return ret;
}


/**
 *Funció necessaria per afegir un node a la llista
 */
int addnode(char *mac, int canal, int utilitzacio,int numeronodes)
{
    int i=0;
    int trobat=0;
      if(strcasecmp(macpropia,mac)==0){
        nodes[0].mac=macpropia;
            nodes[0].canal=canal;
        nodes[0].utilitzacio=utilitzacio;
        trobat=1;
            if(numeronodes==0)
            {
                  return 1;
            }
                return numeronodes;
        }

      while(i<numeronodes)
        {
            if(strcasecmp(nodes[i].mac,mac) == 0){
                        nodes[i].canal=canal;
                nodes[i].utilitzacio=utilitzacio;
                trobat=1;
                return numeronodes;
            }
            i++;
        }
        if(trobat==0)
        {
                  nodes[numeronodes].mac=mac;
            nodes[numeronodes].canal=canal;
            nodes[numeronodes].utilitzacio=utilitzacio;
            return (numeronodes+1);
        }
}

/**
 *Funció per afegir un nou link amb tota la seva informació a la
llista d'enllaós
 */
int addlink(char *mac1,char *mac2,int potenciarebuda,int numerolinks){
   int p=0;
   int find=0;
   while(p<numerolinks){
      if((strcasecmp(enlaces[p].node1mac,mac1)==0)&&(strcasecmp(enlace
s[p].node2mac,mac2)==0)){
            enlaces[p].int21=potenciarebuda;
        find=1;
            return numerolinks;
    }
```

```
if((strcasecmp(enlaces[p].node1mac,mac2)==0)&&(strcasecmp(enlaces[p].n
ode2mac,mac1)==0)){
        enlaces[p].int12=potenciarebuda;
        find=1;
            return numerolinks;
    }
      p++;
  }

  if (find==0){
      strncpy(enlaces[numerolinks].node2mac,mac2,17);
      strncpy(enlaces[numerolinks].node1mac,mac1,17);
    enlaces[numerolinks].int21=potenciarebuda;
    return (numerolinks+1);
 }
}

/**
 *Funció utilitzada per guardar les dades, que s'obtenen mitjançant el
protocolo, en un fitxer extern
*/
void afegirdades(int numeros,int numerol){
      FILE *fitxer;
    fitxer=fopen("/root/lluis/datos","w");
    if (fitxer==NULL){
            fprintf(stderr, "No se puede abrir el fichero de
datos.\n");
        return 1;
    }

      fprintf(fitxer,"%s\n","c tota linia que comenci per c Ã©s un
comentari");
      fprintf(fitxer,"%s\n","c necessaria una linia que comenÃ§a per
n, seguida del numero de nodes");
      fprintf(fitxer,"%s %d\n","n",numeronodes);
      fprintf(fitxer,"%s\n","c linia amb f amb les freq dels nodes
(segons ordre per MAC)");
      fprintf(fitxer,"%s","f ",nodes[0].mac);
      int l=0;

      while (l<numeros){
            fprintf(fitxer,"%d ",nodes[l].canal);
            l++;
      }

      fprintf(fitxer,"\n");
      fprintf(fitxer,"%s\n","c linia amb u utilitzacions dels nodes
(tb segons ordre per MAC)");
    fprintf(fitxer,"%s","u ",nodes[0].mac);
    l=0;

      while (l<numeros){
            fprintf(fitxer,"%d ",nodes[l].utilitzacio);
        l++;
    }

    fprintf(fitxer,"\n");
      fprintf(fitxer,"%s\n","c linies amb e per definir tots els
enllaÃ§os");
```

```
      fprintf(fitxer,"%s\n","c de la forma: e n1 n2 p12 p21; on n1 i
n2 son els id. dels nodes");
      fprintf(fitxer,"%s\n","c segons ordre de MAC p12 i p21 les
potencies de primer node sobre segon i al reves");

      for(l=0;l<numerol;l++){
            fprintf(fitxer,"%s %d %d %d
%d\n","e",returnid(enlaces[l].node1mac,numeros),returnid(enlaces[l].no
de2mac,numeros),enlaces[l].int12,enlaces[l].int21);
      }

      fclose(fitxer);
}

/**
 *Funció que retorna la id d'una MAC per tal de poder mostrar
 *de forma ordenada els valors en el fitxer extern
 */
int returnid(char* mac,int numeroso){
      int q;
      for(q=0;q<numeroso;q++){
            if(strcmp(mac,nodes[q].mac)==0)
                  return q;
      }
}

/**
 *Afegeix un node veí a la llista de nodes
 */
int addnodevei(char *mac,char *macvei, int canal, int
potenciarebuda,int numeronodos)
{
    afegirdades(numeronodos,numerolinks);
      int i=0;
      int provant=0;
    int trobat=0;
    while(i<numeronodes){
            if(strcasecmp(nodes[i].mac,macvei)==0){
                  nodes[i].canal=canal;
            trobat=1;

      numerolinks=addlink(mac,macvei,potenciarebuda,numerolinks);
            return numeronodos;
        }
        i++;
    }
    if(trobat==0){
            numeronodos=addnode(macvei,canal,0,numeronodos);
            numerolinks=addlink(mac,macvei,potenciarebuda,numerolinks);
        return (numeronodos);
    }
}
```

Anexo II. Script de selección de canal

Este script permite que las dos interfaces inalámbricas, elijan el canal que más les convenga, y en el caso de la ad-hoc si tiene algún vecino cerca elige el canal en el que ya se está trabando para que haya conectividad entre ellos.

Cabe mencionar que los datos de sondeo se guardan automáticamente en un archivo log. Desde este archivo se obtienen los datos que se necesitan para rellenar nuestro archivo de información de potencia. Este archivo log para que se actualice cada vez que se hace el sondeo se debe utilizar una función específica: *iwpriv hostscan 1*. De esta forma se consigue tener actualizados los valores continuamente para ser enviados a los demás vecinos.

Código del script de selección de canal:

```
#!/bin/bash

# FUNCION PARA INICIAR EL AP EN EL CANAL HALLADO

recursiu(){
echo "dins"
canal=0
canal_ap=0
aps=(0 0 0 0 0 0 0 0 0 0 0 0 0)
cont=0
linea=1
distancia=0
dist_max=0
dist_izq=0
dist_der=0
cont_aps=0
num_aps=0
canal_ap=""
signal1=0
signal2=0
essid=""
canaladhoc=0
canal_adhoc=0
inicialitzat_adhoc=0

echo "ARA PROVEM AMB EL AP!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
iwpriv wlan0 host_roaming 2 > /dev/null
iwpriv wlan0 host_roaming 2 >> /dev/null

iwpriv wlan0 other_ap_policy 3 > /dev/null
iwpriv wlan0 other_ap_policy 3 >> /dev/null
iwpriv wlan0 hostscan 1

num_aps=$(grep -c ":" < /proc/net/hostap/wlan0/scan_results)

echo "Numero de Access Points encontrados: $num_aps"
echo ""

# OBTENEMOS LA UTILIZACION DEL AP

suma=0
```

```
diferencia=0
suma_anterior=0
carga=0
espera=5
txbytes=0
rxbytes=0
utilizacion=0
umbral=500000

obtener_datos
suma_anterior=$(($txbytes + $rxbytes))
carga=$(($suma_anterior/$espera))
aux=0
        obtener_datos
        suma=$(($txbytes + $rxbytes))
        diferencia=$(($suma - $suma_anterior))
        echo "Diferencia:$diferencia"
        carga=$(($diferencia/$espera))
        echo "Carga:$carga"
        utilizacion=$(($carga/7500 ))
        echo "Utilizacion: $utilizacion"
        echo ""
        suma_anterior=$suma
        sleep $espera
        aux=$(($aux +1))

# ALMACENAR AP'S ENCONTRADOS EN UN VECTOR DE AP'S Y EN EL FICHERO
/ROOT/ALBERT/SCRIPTS/APS Y APS_VECINOS

echo > /root/Albert/scripts/aps


cut -d " " -f7 /proc/net/hostap/wlan0/scan_results  >
/root/Albert/scripts/bssid
cut -d " " -f10 /proc/net/hostap/wlan0/scan_results  >
/root/Albert/scripts/essid
cut -d " " -f1 /proc/net/hostap/wlan0/scan_results  >
/root/Albert/scripts/canal
cut -d " " -f3 /proc/net/hostap/wlan0/scan_results  >
/root/Albert/scripts/signal
cut -d " " -f7 /proc/net/hostap/wlan0/scan_results  >
/root/lluis/dades2
cut -d " " -f1 /proc/net/hostap/wlan0/scan_results  >
/root/lluis/dades2
cut -d " " -f3 /proc/net/hostap/wlan0/scan_results  >
/root/lluis/dades2

echo -n `grep ":" /proc/net/hostap/wlan0/scan_results |cut -d " " -f1`
> /root/Albert/scripts/v_canal

while [ $linea -le $num_aps ];
      do
      canal_ap=$(cut -d " " -f$linea /root/Albert/scripts/v_canal)
      aux=$((canal_ap-1))
      echo "valor de aux $aux"
      if [ $aux -lt 0 ]
      then
      aux=0
      fi
            aps[$aux]=1
            linea=$(($linea+1))
```

```
done

echo "OCUPACION DE CANALES -->" ${aps[@]}
echo ""
# Almacenar los datos de los APs en ficheros

line=1
iwconfig wlan0 > /root/Albert/scripts/iwconfig_res
essid=$(grep "ESSID" /root/Albert/scripts/iwconfig_res | cut -d ":" -
f2 | cut -d " " -f1)
envio=$essid";"
bssid=$(ifconfig |grep "wlan0"  | cut -d " " -f10)
envio=$envio$bssid";"
freq=$(grep "Access Point" /root/Albert/scripts/iwconfig_res | cut -d
" " -f13 | cut -d ":" -f2)
envio=$envio$freq";"

echo -e  "Utilitzacio" > utilit
rm  /root/Albert/scripts/aps
rm /root/Albert/scripts/aps_vecinos
echo "$utilizacion $bssid $canal_hostap " >> /root/Albert/scripts/aps

paste -d " " bssid essid canal signal >
/root/Albert/scripts/aps_vecinos
grep -m $num_aps ":" /root/Albert/scripts/aps_vecinos
>>/root/Albert/scripts/aps

rm -f aps_vecinos

echo "BSSID ESSID CANAL SIGNAL" > aps_vecinos

paste -d " " bssid essid canal signal >>
/root/Albert/scripts/aps_vecinos2

grep -m $num_aps "hostap" /root/Albert/scripts/aps_vecinos2 >>
/root/Albert/scripts/aps_vecinos
# Datos Guardados, borrar ficheros no utilizados

rm -f aps_vecinos2
rm -f bssid
rm -f essid
rm -f canal
rm -f signal
rm -f v_canal
rm -f u
rm -f utilit
rm -f iwconfig_res
rm -f /root/lluis/dades2


# vaciado del archivo de texto donde se imprimirá la distancia máxima
para tratarla

echo -n `rm -f /root/scripts_oscar/distancia_max`
echo -n `touch /root/scripts_oscar/distancia_max`

# echo "ESCANEO DE CANALES"
# Calculo de distancias entre los APs recorriendo cada canal

while [ $cont -lt 13 ];
do
```

```
   # evaluación de si hay AP (1) o no hay AP (0) en cada canal

   if [ $((aps[$cont])) -eq 0 ]
   then
        # caso en que se llega al extremo de la derecha
        if [ $cont -eq 12 ]
        then
           # Cálculo de la distancia al extremo derecho

           distancia=$(($distancia+1))
           dist_der=$((distancia))
           echo " "
           echo "Canal $(($cont+1)): no hay AP. Ultimo canal"
           echo "La distancia al extremo derecho es $dist_der entre
canales $(($cont+1)) y $(($cont - $((dist_der-1))))"
           echo "La distancia al extremo derecho es $dist_der entre
canales $(($cont+1)) y $(($cont - $((dist_der-1))))" >>
/root/scripts_oscar/distancia_max
           echo " "
        else
           distancia=$(($distancia+1))
           echo "Canal $(($cont+1)): no hay AP"
        fi
   else
           cont_aps=$(($cont_aps+1))

         if [ $cont_aps -eq 1 ]
           then

              dist_izq=$((distancia))
              distancia=0

              echo " "
              echo "Canal $(($cont+1)): hay AP"
              echo "La distancia al extremo izquierdo es $dist_izq
entre canales $(($cont+1)) y $(($(($cont+1)) - $dist_izq))"
              echo "La distancia al extremo izquierdo es $dist_izq
entre canales $(($cont+1)) y $(($(($cont+1)) - $dist_izq))" >>
/root/scripts_oscar/distancia_max
              echo " "

           else

              if [ $cont -eq 12 ]
              then

                 # Cálculo de la distancia al extremo derecho

                 echo " "
                 echo "Canal $(($cont+1)): hay AP. Ultimo canal"

                 if [ $distancia -gt $dist_max ]
                 then
                    dist_max=$((distancia))
                    distancia=0
                    dist_der=0
                    echo "La distancia max es $dist_max entre canales
$(($cont+1)) y $(($cont - $dist_max))"
                    echo "La distancia max es $dist_max entre canales
$(($cont+1)) y $(($cont - $dist_max))" >>
/root/scripts_oscar/distancia_max
```

```
                fi

                echo "La distancia al extremo derecho es $dist_der
entre canales $(($cont+1)) y $(($((cont+1)) - $dist_der))"
                echo "La distancia al extremo derecho es $dist_der
entre canales $(($cont+1)) y $(($((cont+1)) - $dist_der))" >>
/root/scripts_oscar/distancia_max
                echo " "

             else

                # actualización o no de dist_max si hay AP

             if [ $distancia -gt $dist_max ]
                then
                   dist_max=$((distancia))
                   distancia=0
                   echo " "
                   echo "Canal $(($cont+1)): hay AP"
                   echo "La distancia max es $dist_max entre canales
$(($cont+1)) y $(($cont - $dist_max))"
                   echo "La distancia max es $dist_max entre canales
$(($cont+1)) y $(($cont - $dist_max))" >>
/root/scripts_oscar/distancia_max
                   echo " "
                else
                   distancia=0
                   echo " "
                   echo "Canal $(($cont+1)): hay AP"
                   echo $(grep "max"
/root/scripts_oscar/distancia_max)
                   echo " "
                fi
             fi
          fi
   fi
   cont=$((cont+1))
done

# Evaluamos si distancia maxima es mayor que 2 veces el estandar

if [ $dist_max -gt $((2*4)) ]
then
   canal1=$(grep "max" /root/scripts_oscar/distancia_max | cut -d " "
-f8)
   canal2=$(grep "max" /root/scripts_oscar/distancia_max | cut -d " "
-f10)
   echo " "
   echo "$canal1"
   echo "$canal2"
   echo " "
   canal_hostap=$(($(($canal1+$canal2))/2))
   echo "El hostAP trabajara en el canal $canal_hostap"
   colocar_ap
   aps[$(($canal_hostap-1))]=X


elif [ $dist_izq -gt 4 -a $dist_der -gt 4 ]
then
   if [ $dist_izq -ge $dist_der ]
then
```

```
      canal_hostap=1
      echo " "
      echo "El hostAP trabajara en el canal $canal_hostap"
      colocar_ap
      aps[$(($canal_hostap-1))]=X
   else
      canal_hostap=13
      echo " "
      echo "El hostAP trabajara en el canal $canal_hostap"
      colocar_ap
      aps[$(($canal_hostap-1))]=X
   fi

elif [ $dist_izq -gt 4 -a $dist_der -le 4 ]
then
   canal_hostap=1
   echo " "
   echo "El hostAP trabajara en el canal $canal_hostap"
   colocar_ap
   aps[$(($canal_hostap-1))]=X

elif [ $dist_der -gt 4 -a $dist_izq -le 4 ]
then
   canal_hostap=13
   echo " "
   echo "El hostAP trabajara en el canal $canal_hostap"
   colocar_ap
   aps[$(($canal_hostap-1))]=X

# Eleccion segun la potencia emitida por cada AP

else

   canal1=$(grep "max" /root/scripts_oscar/distancia_max | cut -d " "
-f10)
   canal2=$(grep "max" /root/scripts_oscar/distancia_max | cut -d " "
-f8)
   echo "$canal1 $canal2"
   snr1=$(grep "$canal1" /proc/net/hostap/wlan0/scan_results | cut -d
" " -f2)
   snr2=$(grep "$canal2" /proc/net/hostap/wlan0/scan_results | cut -d
" " -f2)
   echo "SNR del AP del canal $canal1: $snr1"
   echo "SNR del AP del canal $canal2: $snr2"


if [ $(($dist_max % 2)) != 0 ]
   then
      if [ $(($snr1-$snr2)) -lt 15 -o $(($snr2-$snr1)) -gt -15 ]
      then
         canal_hostap=$(($(($canal1+$canal2))/2))
         echo " "
         echo "caso 1: El hostAP trabajará en el canal $canal_hostap"
         colocar_ap
         aps[$(($canal_hostap-1))]=X
      elif [ $snr1 -gt $snr2 ]
      then
         canal_hostap=$(($(($(($canal1+$canal2))/2))+1))
         echo " "
         echo "caso 2: El hostAP trabajara en el canal $canal_hostap"
         colocar_ap
```

```
         aps[$(($canal_hostap-1))]=X
      else
         canal_hostap=$(($(($(($canal1+$canal2))/2))-1))
         echo " "
         echo "caso 3: El hostAP trabajara en el canal $canal_hostap"
         colocar_ap
         aps[$(($canal_hostap-1))]=X
      fi
   elif [ $snr1 -gt $snr2 ]
   then
      canal_hostap=$(($(($(($canal1+$canal2))/2))+1))
      echo " "
      echo "caso 4: El hostAP trabajara en el canal $canal_hostap"
      colocar_ap
      aps[$(($canal_hostap-1))]=X
   else
      canal_hostap=$(($(($canal1+$canal2))/2))
      echo " "
      echo "caso 5: El hostAP trabajara en el canal $canal_hostap"
      colocar_ap
      if [ $(($canal_hostap-1)) -lt 12 ]
      then
            aps[$(($canal_hostap-1))]=X
      fi
   fi
fi

echo " "
echo "OCUPACION DE CANALES -->" ${aps[@]}
echo " "

while [ $line -le $num_aps ];
do
   if [ $line -eq $num_aps ]
   then
      otros=$(grep "bssid" -m $line
/proc/net/hostap/wlan0/scan_results | cut -d " " -f3 | cut -d "=" -f2
| tail -n 1)
      envio=$envio$otros";A-"
      line=$((line+1))
   else
      otros=$(grep "bssid" -m $line
/proc/net/hostap/wlan0/scan_results | cut -d " " -f3 | cut -d "=" -f2
| tail -n 1)
      envio=$envio$otros"--"
      line=$((line+1))
   fi

done

echo -n $envio > /root/scripts_oscar/string_anuncio

/root/Albert/scripts/fichero_carga &
}

colocar_ap()
{
      iwconfig wlan0 essid hostap 2>/dev/null
      iwconfig wlan0 mode master 2>/dev/null
      iwconfig wlan0 channel $canal_hostap 2>/dev/null
      echo "HostAP iniciado en modo Master en el canal $canal_hostap"
```

```
}

colocar_adhoc()
{
      iwconfig eth1 essid wmn 2>/dev/null
      iwconfig eth1 mode ad-hoc 2>/dev/null
      iwconfig eth1 channel $canal_adhoc 2>/dev/null
      echo "Tarjeta de control iniciada en modo Ad-hoc en el canal
$canal_adhoc"
      sleep 20
}

obtener_datos ()
{
        iwpriv wlan0 inquire 61696
        txbytes=$(ifconfig wlan0 | grep "RX bytes" |cut -d " " -f12|
cut -d ":" -f2)
        rxbytes=$(ifconfig wlan0 | grep "RX bytes" |cut -d " " -f17|
cut -d ":" -f2)
}

canal=0
canal_ap=0
aps=(0 0 0 0 0 0 0 0 0 0 0 0 0)
cont=0
linea=1
distancia=0
dist_max=0
dist_izq=0
dist_der=0
cont_aps=0
num_aps=0
canal_ap=""
signal1=0
signal2=0
essid=""
canaladhoc=0
canal_adhoc=0
inicialitzat_adhoc=0

#INICIAMOS LA INTERFAZ INALAMBRICA EN MODO MANAGED
iwconfig wlan0 mode managed 2>/dev/null
rm -f /root/Albert/scripts/aps
#clear

#COMANDOS PARA PROVOCAR EL ESCANEO MANUAL Y LA RECEPCION DE TODOS LOS
BEACONS

contadoradhoc=1
essidadhoc="wmn"

iwpriv wlan0 host_roaming 2 > /dev/null
iwpriv wlan0 host_roaming 2 >> /dev/null

iwpriv wlan0 other_ap_policy 3 > /dev/null
iwpriv wlan0 other_ap_policy 3 >> /dev/null
iwlist wlan0 scan
num_aps=$(grep -c ":" < /proc/net/hostap/wlan0/scan_results)

echo "Numero de Access Points encontrados: $num_aps"
echo ""
```

```
while [ $contadoradhoc -lt $num_aps ]
do
      essid=$(echo -n `grep ":" /proc/net/hostap/wlan0/scan_results
|cut -d " " -f10` |cut -d " " -f$contadoradhoc)
      canaladhoc=$(echo -n `grep ":"
/proc/net/hostap/wlan0/scan_results |cut -d " " -f1` |cut -d " " -
f$contadoradhoc)
      echo $essid
      case $essid in
      $essidadhoc)
            echo "Hem trobat un punt ad-hoc proper de la nostra xarxa
accedim al mateix canal"
            iwconfig eth1 mode ad-hoc essid wmn channel $canaladhoc
            sleep 20
            inicialitzat_adhoc=1
            ;;
      esac
      contadoradhoc=$(($contadoradhoc+1))
done

if [ $inicialitzat_adhoc -eq 0 ]
then
# OBTENEMOS LA UTILIZACION DEL AP
echo "entramos en la busqueda del canal!!!!!!!!!!!!!!!!"

suma=0
diferencia=0
suma_anterior=0
carga=0
espera=5
txbytes=0
rxbytes=0
utilizacion=0
umbral=500000


obtener_datos
suma_anterior=$(($txbytes + $rxbytes))
carga=$(($suma_anterior/$espera))
aux=0


        obtener_datos
        suma=$(($txbytes + $rxbytes))
        diferencia=$(($suma - $suma_anterior))
        echo "Diferencia:$diferencia"
        carga=$(($diferencia/$espera))
        echo "Carga:$carga"
        utilizacion=$(($carga/7500 ))
        echo "Utilizacion: $utilizacion"
        echo ""
        suma_anterior=$suma
        sleep $espera
        aux=$(($aux +1))

obtener_datos

# ALMACENAR AP'S ENCONTRADOS EN UN VECTOR DE AP'S Y EN EL FICHERO
/ROOT/ALBERT/SCRIPTS/APS Y APS_VECINOS
```

```
echo ""     > /root/Albert/scripts/aps

echo "estem aqui $num_aps"
cut -d " " -f7 /proc/net/hostap/wlan0/scan_results  >
/root/Albert/scripts/bssid
cut -d " " -f7 /proc/net/hostap/wlan0/scan_results  >
/root/lluis/dades2
cut -d " " -f10 /proc/net/hostap/wlan0/scan_results  >
/root/Albert/scripts/essid
cut -d " " -f1 /proc/net/hostap/wlan0/scan_results  >
/root/Albert/scripts/canal
cut -d " " -f1 /proc/net/hostap/wlan0/scan_results  >
/root/lluis/dades2
cut -d " " -f3 /proc/net/hostap/wlan0/scan_results  >
/root/Albert/scripts/signal
cut -d " " -f3 /proc/net/hostap/wlan0/scan_results  >
/root/lluis/dades2
echo -n `grep ":" /proc/net/hostap/wlan0/scan_results |cut -d " " -f1`
> /root/Albert/scripts/v_canal

while [ $linea -le $num_aps ];
      do
      canal_ap=$(cut -d " " -f$linea /root/Albert/scripts/v_canal)
      echo "valor de aux $aux"

      aux=$((canal_ap-1))
      echo "valor de aux $aux"
            aps[$aux]=1
            linea=$(($linea+1))
done

echo "OCUPACION DE CANALES -->" ${aps[@]}
echo ""
# Almacenar los datos de los APs en ficheros

line=1
iwconfig wlan0 > /root/Albert/scripts/iwconfig_res
essid=$(grep "ESSID" /root/Albert/scripts/iwconfig_res | cut -d ":" -
f2 | cut -d " " -f1)
envio=$essid";"
bssid=$(ifconfig |grep "wlan0"  | cut -d " " -f10)
envio=$envio$bssid";"
freq=$(grep "Access Point" /root/Albert/scripts/iwconfig_res | cut -d
" " -f13 | cut -d ":" -f2)
envio=$envio$freq";"

echo -e  "Utilitzacio" > utilit

echo "$utilizacion $bssid $essid $canal_adhoc 0" >>
/root/Albert/scripts/aps

#paste -d " " $bssid essid canal signal >
/root/Albert/scripts/aps_vecinos
#tail -n $num_aps /root/Albert/scripts/aps_vecinos >>
/root/Albert/scripts/aps
grep -m $num_aps ":" /root/Albert/scripts/aps_vecinos
>>/root/Albert/scripts/aps

rm -f aps_vecinos

echo "BSSID ESSID CANAL SIGNAL" > aps_vecinos
```

```
#paste -d " " bssid essid canal signal >>
/root/Albert/scripts/aps_vecinos2

grep -m $num_aps "hostap" /root/Albert/scripts/aps_vecinos2 >>
/root/Albert/scripts/aps_vecinos

# Datos Guardados, borrar ficheros no utilizados

rm -f aps_vecinos2
rm -f bssid
rm -f essid
rm -f canal
rm -f signal
rm -f v_canal
rm -f u
rm -f utilit
rm -f iwconfig_res

# vaciado del archivo de texto donde se imprimirá la distancia máxima
para tratarla

echo -n `rm -f /root/scripts_oscar/distancia_max`
echo -n `touch /root/scripts_oscar/distancia_max`

# echo "ESCANEO DE CANALES"
# Calculo de distancias entre los APs recorriendo cada canal

while [ $cont -lt 13 ];
do
   # evaluación de si hay AP (1) o no hay AP (0) en cada canal
   if [ $((aps[$cont])) -eq 0 ]
   then
        # caso en que se llega al extremo de la derecha
        if [ $cont -eq 12 ]
        then
           # Cálculo de la distancia al extremo derecho
           distancia=$(($distancia+1))
           dist_der=$((distancia))
           echo " "
           echo "Canal $(($cont+1)): no hay AP. Ultimo canal"
           echo "La distancia al extremo derecho es $dist_der entre
canales $(($cont+1)) y $(($cont - $((dist_der-1))))"
           echo "La distancia al extremo derecho es $dist_der entre
canales $(($cont+1)) y $(($cont - $((dist_der-1))))" >>
/root/scripts_oscar/distancia_max
           echo " "
        else
           distancia=$(($distancia+1))
           echo "Canal $(($cont+1)): no hay AP"
        fi
   else
           cont_aps=$(($cont_aps+1))
           # Distancia al extremo izquierdo
      if [ $cont_aps -eq 1 ]
           then

              dist_izq=$((distancia))
              distancia=0

              echo " "
```

```
              echo "Canal $(($cont+1)): hay AP"
              echo "La distancia al extremo izquierdo es $dist_izq
entre canales $(($cont+1)) y $(($(($cont+1)) - $dist_izq))"
              echo "La distancia al extremo izquierdo es $dist_izq
entre canales $(($cont+1)) y $(($(($cont+1)) - $dist_izq))" >>
/root/scripts_oscar/distancia_max
              echo " "

           else

              if [ $cont -eq 12 ]
              then

                 # Cálculo de la distancia al extremo derecho

                 echo " "
                 echo "Canal $(($cont+1)): hay AP. Ultimo canal"

                 if [ $distancia -gt $dist_max ]
                 then
                    dist_max=$((distancia))
                    distancia=0
                    dist_der=0
                    echo "La distancia max es $dist_max entre canales
$(($cont+1)) y $(($cont - $dist_max))"
                    echo "La distancia max es $dist_max entre canales
$(($cont+1)) y $(($cont - $dist_max))" >>
/root/scripts_oscar/distancia_max
                 fi

                 echo "La distancia al extremo derecho es $dist_der
entre canales $(($cont+1)) y $(($((cont+1)) - $dist_der))"
                 echo "La distancia al extremo derecho es $dist_der
entre canales $(($cont+1)) y $(($((cont+1)) - $dist_der))" >>
/root/scripts_oscar/distancia_max
                 echo " "

              else

                 # actualización o no de dist_max si hay AP

if [ $distancia -gt $dist_max ]
                 then
                    dist_max=$((distancia))
                    distancia=0
                    echo " "
                    echo "Canal $(($cont+1)): hay AP"
                    echo "La distancia max es $dist_max entre canales
$(($cont+1)) y $(($cont - $dist_max))"
                    echo "La distancia max es $dist_max entre canales
$(($cont+1)) y $(($cont - $dist_max))" >>
/root/scripts_oscar/distancia_max
                    echo " "
                 else
                    distancia=0
                    echo " "
                    echo "Canal $(($cont+1)): hay AP"
                    echo $(grep "max"
/root/scripts_oscar/distancia_max)
                    echo " "
                 fi
```

```
            fi
         fi
   fi
   cont=$((cont+1))
done

# Evaluamos si distancia maxima es mayor que 2 veces el estandar

if [ $dist_max -gt $((2*4)) ]
then
   canal1=$(grep "max" /root/scripts_oscar/distancia_max | cut -d " "
-f8)
   canal2=$(grep "max" /root/scripts_oscar/distancia_max | cut -d " "
-f10)
   echo " "
   echo "$canal1"
   echo "$canal2"
   echo " "
   canal_adhoc=$(($(($canal1+$canal2))/2))
   echo "La tarjeta ad-hoc trabajara en el canal $canal_adhoc"
   colocar_adhoc
   aps[$(($canal_adhoc-1))]=X


elif [ $dist_izq -gt 4 -a $dist_der -gt 4 ]
then
   if [ $dist_izq -ge $dist_der ]
      then
      canal_adhoc=1
      echo " "
      echo "La tarjeta ad-hoc trabajara en el canal $canal_adhoc"
      colocar_adhoc
      aps[$(($canal_adhoc-1))]=X
   else
      canal_adhoc=13
      echo " "
      echo "La tarjeta ad-hoc trabajara en el canal $canal_adhoc"
      colocar_adhoc
      aps[$(($canal_adhoc-1))]=X
   fi

elif [ $dist_izq -gt 4 -a $dist_der -le 4 ]
then
   canal_adhoc=1
   echo " "
   echo "La tarjeta ad-hoc trabajara en el canal $canal_adhoc"
   colocar_adhoc
   aps[$(($canal_adhoc-1))]=X

elif [ $dist_der -gt 4 -a $dist_izq -le 4 ]
then
   canal_adhoc=13
   echo " "
   echo "la tarjeta ad-hoc trabajara en el canal $canal_adhoc"
   colocar_adhoc
   aps[$(($canal_adhoc-1))]=X

# Eleccion segun la potencia emitida por cada AP

else
```

```
   canal1=$(grep "max" /root/scripts_oscar/distancia_max | cut -d " "
-f10)
   canal2=$(grep "max" /root/scripts_oscar/distancia_max | cut -d " "
-f8)
   echo "$canal1 $canal2"
   snr1=$(grep "$canal1" /proc/net/hostap/wlan0/scan_results | cut -d
" " -f2)
   snr2=$(grep "$canal2" /proc/net/hostap/wlan0/scan_results | cut -d
" " -f2)

   echo "SNR del AP del canal $canal1: $snr1"
   echo "SNR del AP del canal $canal2: $snr2"


if [ $(($dist_max % 2)) != 0 ]
   then
      if [ $(($snr1-$snr2)) -lt 15 -o $(($snr2-$snr1)) -gt -15 ]
      then
         canal_adhoc=$(($(($canal1+$canal2))/2))
         echo " "
         echo "caso 1: LA tarjeta ad-hoc trabajará en el canal
$canal_adhoc"
         colocar_adhoc
         aps[$(($canal_adhoc-1))]=X
      elif [ $snr1 -gt $snr2 ]
      then
         canal_adhoc=$(($(($(($canal1+$canal2))/2))+1))
         echo " "
         echo "caso 2: La tarjeta ad-hoc trabajara en el canal
$canal_adhoc"
         colocar_adhoc
         aps[$(($canal_adhoc-1))]=X
      else
         canal_adhoc=$(($(($(($canal1+$canal2))/2))-1))
         echo " "
         echo "caso 3: La tarjeta ad-hoc trabajara en el canal
$canal_adhoc"
         colocar_adhoc
         aps[$(($canal_adhoc-1))]=X
      fi
   elif [ $snr1 -gt $snr2 ]
   then
      canal_adhoc=$(($(($(($canal1+$canal2))/2))+1))
      echo " "
      echo "caso 4: La tarjeta ad-hoc trabajara en el canal
$canal_adhoc"
      colocar_adhoc
      aps[$(($canal_adhoc-1))]=X
   else
      canal_adhoc=$(($(($canal1+$canal2))/2))
      echo " "
      echo "caso 5: La tarjeta ad-hoc trabajara en el canal
$canal_adhoc"
      colocar_adhoc
      aps[$(($canal_adhoc-1))]=X
   fi
fi

echo " "
echo "OCUPACION DE CANALES -->" ${aps[@]}
echo " "
```

```
while [ $line -le $num_aps ];
do
   if [ $line -eq $num_aps ]
   then
      otros=$(grep "bssid" -m $line
/proc/net/hostap/wlan0/scan_results | cut -d " " -f3 | cut -d "=" -f2
| tail -n 1)
      envio=$envio$otros";A-"
      line=$((line+1))
   else
      otros=$(grep "bssid" -m $line
/proc/net/hostap/wlan0/scan_results | cut -d " " -f3 | cut -d "=" -f2
| tail -n 1)
      envio=$envio$otros"--"
      line=$((line+1))
   fi
done
fi

while [ 1 ];
do
echo " entrem un cop"
recursiu
done
```